IBM MQ

Administering IBM MQ

Version 8 Release 0

Note

Before using this information and the product it supports, read the information in "Notices" on page 1897.

This edition applies to version 8 release 0 of WebSphere MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

Administering IBM MQ

Administering queue managers and associated resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your queue managers and associated resources.

You can administer IBM® MQ objects locally or remotely, see “Local and remote administration” on page 4.

There are a number of different methods that you can use to create and administer your queue managers and their related resources in IBM MQ. These methods include command-line interfaces, a graphical user interface, and an administration API. See the sections and links in this topic for more information about each of these interfaces.

There are different sets of commands that you can use to administer IBM MQ depending on your platform:
- “IBM MQ control commands”
- “IBM MQ Script (MQSC) commands”
- “Programmable Command Formats (PCFs)” on page 2
- IBM i “IBM i Control Language (CL)” on page 3

There are also the other following options for creating and managing IBM MQ objects:
- “The MQ Explorer” on page 3
- “The Windows Default Configuration application” on page 3
- “The Microsoft Cluster Service (MSCS)” on page 3

z/OS For information about the administration interfaces and options on IBM MQ for z/OS®, see “Administering IBM MQ for z/OS” on page 255.

You can automate some administration and monitoring tasks for both local and remote queue managers by using PCF commands. These commands can also be simplified through the use of the IBM MQ Administration Interface (MQAI) on some platforms. For more information about automating administration tasks, see “Automating administration tasks” on page 5.

IBM MQ control commands

Control commands allow you to perform administrative tasks on queue managers themselves.

IBM MQ for Windows, UNIX and Linux systems provides the *control commands* that you issue at the system command line.

The control commands are described in Creating and managing queue managers on distributed platforms. For the command reference for the control commands, see IBM MQ Control commands.

IBM MQ Script (MQSC) commands

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

You issue MQSC commands to a queue manager by using the **runmqsc** command. You can do this interactively, issuing commands from a keyboard, or you can redirect the standard input device (stdin) to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same.

You can run the **runmqsc** command in three modes, depending on the flags set on the command:
- *Verification mode*, where the MQSC commands are verified on a local queue manager, but are not run
- *Direct mode*, where the MQSC commands are run on a local queue manager
- *Indirect mode*, where the MQSC commands are run on a remote queue manager

Object attributes specified in MQSC commands are shown in this section in uppercase (for example, RQMNAME), although they are not case-sensitive. MQSC command attribute names are limited to eight characters.

MQSC commands are available on all platforms z/OS IBM i , including IBM i, and z/OS . MQSC commands are summarized in Comparing command sets.

On Windows, UNIX or Linux, you can use the MQSC as single commands issued at the system command line. To issue more complicated, or multiple commands, the MQSC can be built into a file that you run from the Windows, UNIX or Linux system command line. MQSC can be sent to a remote queue manager. For full details, see Building command scripts.

IBM i On IBM i, to issue the commands on an IBM i server, create a list of commands in a Script file, and then run the file by using the STRMQMMQSC command.

Notes: IBM i
1. Do not use the QTEMP library as the input library to STRMQMMQSC, as the usage of the QTEMP library is limited. You must use another library as an input file to the command.
2. On IBM i, MQSC responses to commands issued from a script file are returned in a spool file.

"Script (MQSC) Commands" on page 74 contains a description of each MQSC command and its syntax.

See "Performing local administration tasks using MQSC commands" on page 73 for more information about using MQSC commands in local administration.

Programmable Command Formats (PCFs)

Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. You can use PCF commands in a systems management application program for administration of IBM MQ objects: authentication information objects, channels, channel listeners, namelists, process definitions, queue managers, queues, services, and storage classes. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local, or remote, using the local queue manager.

For more information about PCFs, see "Introduction to Programmable Command Formats" on page 6.

For definition of PCFs and structures for the commands and responses, see Programmable command formats reference.

IBM i

IBM i Control Language (CL)

This language can be used to issue administration commands to IBM MQ for IBM i. The commands can be issued either at the command line or by writing a CL program. These commands perform similar functions to PCF commands, but the format is different. CL commands are designed exclusively for servers and CL responses are designed to be human-readable, whereas PCF commands are platform independent and both command and response formats are intended for program use.

For full details of the IBM i Control Language (CL), see IBM MQ for IBM i CL commands.

The MQ Explorer

Using the MQ Explorer, you can perform the following actions:
- Define and control various resources including queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and clusters.
- Start or stop a local queue manager and its associated processes.
- View queue managers and their associated objects on your workstation or from other workstations.
- Check the status of queue managers, clusters, and channels.
- Check to see which applications, users, or channels have a particular queue open, from the queue status.

On Windows and Linux systems, you can start MQ Explorer by using the system menu, the `MQExplorer` executable file, or the **`strmqcfg`** command.

On Linux, to start the MQ Explorer successfully, you must be able to write a file to your home directory, and the home directory must exist.

See “Administration using the MQ Explorer” on page 56 for more information.

You can use MQ Explorer to administer remote queue managers on other platforms including z/OS, for details and to download the SupportPac MS0T, see http://www.ibm.com/support/docview.wss?uid=swg24021041.

The Windows Default Configuration application

You can use the Windows Default Configuration program to create a *starter* (or default) set of IBM MQ objects. A summary of the default objects created is listed in Table 1: Objects created by the Windows default configuration application.

The Microsoft Cluster Service (MSCS)

Microsoft Cluster Service (MSCS) enables you to connect servers into a *cluster*, giving higher availability of data and applications, and making it easier to manage the system. MSCS can automatically detect and recover from server or application failures.

It is important not to confuse clusters in the MSCS sense with IBM MQ clusters. The distinction is:

IBM MQ clusters
are groups of two or more queue managers on one or more computers, providing automatic interconnection, and allowing queues to be shared among them for load balancing and redundancy.

MSCS clusters
Groups of computers, connected together and configured in such a way that, if one fails, MSCS performs a *failover*, transferring the state data of applications from the failing computer to another computer in the cluster and reinitiating their operation there.

Supporting the Microsoft Cluster Service (MSCS) provides detailed information about how to configure your IBM MQ for Windows system to use MSCS.

Related concepts:

"Administering local IBM MQ objects" on page 69
This section tells you how to administer local IBM MQ objects to support application programs that use the Message Queue Interface (MQI). In this context, local administration means creating, displaying, changing, copying, and deleting IBM MQ objects.

"Administering remote IBM MQ objects" on page 129

IBM i "Administering IBM i" on page 183
Introduces the methods available to you to administer IBM MQ on IBM i.

z/OS "Administering IBM MQ for z/OS" on page 255
Administering queue managers and associated resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your queue managers and associated resources.

Related information:

IBM MQ technical overview

Planning

z/OS Planning your IBM MQ environment on z/OS

Configuring

z/OS Configuring z/OS

Transactional support scenarios

Considerations when contact is lost with the XA resource manager

Local and remote administration

You can administer IBM MQ objects locally or remotely.

Local administration means carrying out administration tasks on any queue managers you have defined on your local system. You can access other systems, for example through the TCP/IP terminal emulation program **telnet**, and carry out administration there. In IBM MQ, you can consider this as local administration because no channels are involved, that is, the communication is managed by the operating system.

IBM MQ supports administration from a single point of contact through what is known as *remote administration*. This allows you to issue commands from your local system that are processed on another system and applies also to the IBM MQ Explorer. For example, you can issue a remote command to change a queue definition on a remote queue manager. You do not have to log on to that system, although you do need to have the appropriate channels defined. The queue manager and command server on the target system must be running.

Some commands cannot be issued in this way, in particular, creating or starting queue managers and starting command servers. To perform this type of task, you must either log onto the remote system and issue the commands from there or create a process that can issue the commands for you. This restriction applies also to the IBM MQ Explorer.

"Administering remote IBM MQ objects" on page 129 describes the subject of remote administration in greater detail.

How to use IBM MQ control commands

This section describes how to use the IBM MQ control commands.

If you want to issue control commands, your user ID must be a member of the mqm group. For more information about this, see Authority to administer IBM MQ on UNIX, Linux and Windows systems. In addition, note the following environment-specific information:

IBM MQ for Windows

All control commands can be issued from a command line. Command names and their flags are not case sensitive: you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase. However, arguments to control commands (such as queue names) are case sensitive.

In the syntax descriptions, the hyphen (-) is used as a flag indicator. You can use the forward slash (/) instead of the hyphen.

IBM MQ for UNIX and Linux systems

All IBM MQ control commands can be issued from a shell. All commands are case-sensitive.

A subset of the control commands can be issued using the IBM MQ Explorer.

For more information, see The IBM MQ control commands

Automating administration tasks

You might decide that it would be beneficial to your installation to automate some administration and monitoring tasks. You can automate administration tasks for both local and remote queue managers using programmable command format (PCF) commands. This section assumes that you have experience of administering IBM MQ objects.

PCF commands

IBM MQ programmable command format (PCF) commands can be used to program administration tasks into an administration program. In this way, from a program you can manipulate queue manager objects (queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects), and even manipulate the queue managers themselves.

PCF commands cover the same range of functions provided by MQSC commands. You can write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of an IBM MQ message. Each command is sent to the target queue manager using the MQI function **MQPUT** in the same way as any other message. Providing the command server is running on the queue manager receiving the message, the command server interprets it as a command message and runs the command. To get the replies, the application issues an **MQGET** call and the reply data is returned in another data structure. The application can then process the reply and act accordingly.

Note: Unlike MQSC commands, PCF commands and their replies are not in a text format that you can read.

Briefly, these are some of the things needed to create a PCF command message:

Message descriptor

This is a standard IBM MQ message descriptor, in which:

- Message type (`MsqType`) is MQMT_REQUEST.

- Message format (*Format*) is MQFMT_ADMIN.

Application data
Contains the PCF message including the PCF header, in which:
- The PCF message type (*Type*) specifies MQCFT_COMMAND.
- The command identifier specifies the command, for example, *Change Queue* (MQCMD_CHANGE_Q).

For a complete description of the PCF data structures and how to implement them, see “Introduction to Programmable Command Formats.”

PCF object attributes

Object attributes in PCF are not limited to eight characters as they are for MQSC commands. They are shown in this guide in italics. For example, the PCF equivalent of RQMNAME is *RemoteQMgrName*.

Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about escape PCFs, see Escape.

Introduction to Programmable Command Formats

Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. PCFs simplify queue manager administration and other network administration. They can be used to solve the problem of complex administration of distributed networks especially as networks grow in size and complexity.

The Programmable Command Formats described in this product documentation are supported by:
- IBM MQ for AIX®
- IBM MQ for HP-UX
- IBM i IBM MQ for IBM i
- IBM MQ for Linux
- IBM MQ for Solaris
- IBM MQ for Windows
- z/OS IBM MQ for z/OS
- IBM IBM WebSphere® MQ for HP Integrity NonStop Server V5.3

The problem PCF commands solve

The administration of distributed networks can become complex. The problems of administration continue to grow as networks increase in size and complexity.

Examples of administration specific to messaging and queuing include:
- Resource management.

  For example, queue creation and deletion.
- Performance monitoring.

  For example, maximum queue depth or message rate.
- Control.

  For example, tuning queue parameters such as maximum queue depth, maximum message length, and enabling and disabling queues.

- Message routing.

  Definition of alternative routes through a network.

IBM MQ PCF commands can be used to simplify queue manager administration and other network administration. PCF commands allow you to use a single application to perform network administration from a single queue manager within the network.

What are PCFs?

PCFs define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. You can use PCF commands in a systems management application program for administration of IBM MQ objects: authentication information objects, channels, channel listeners, namelists, process definitions, queue managers, queues, services, and storage classes. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local, or remote, using the local queue manager.

Each queue manager has an administration queue with a standard queue name and your application can send PCF command messages to that queue. Each queue manager also has a command server to service the command messages from the administration queue. PCF command messages can therefore be processed by any queue manager in the network and the reply data can be returned to your application, using your specified reply queue. PCF commands and reply messages are sent and received using the normal Message Queue Interface (MQI).

For a list of the available PCF commands, including their parameters, see Definitions of the Programmable Command Formats.

Using Programmable Command Formats

You can use PCFs in a systems management program for IBM MQ remote administration.

This section includes:

- “PCF command messages”
- “Responses” on page 10
- z/OS “Extended responses” on page 12
- Rules for naming IBM MQ objects
- “Authority checking for PCF commands” on page 14

PCF command messages:

PCF command messages consist of a PCF header, parameters identified in that header and also user-defined message data. The messages are issued using Message Queue interface calls.

Each command and its parameters are sent as a separate command message containing a PCF header followed by a number of parameter structures; for details of the PCF header, see MQCFH - PCF header, and for an example of a parameter structure, see MQCFST - PCF string parameter. The PCF header identifies the command and the number of parameter structures that follow in the same message. Each parameter structure provides a parameter to the command.

Replies to the commands, generated by the command server, have a similar structure. There is a PCF header, followed by a number of parameter structures. Replies can consist of more than one message but commands always consist of one message only.

On platforms other than z/OS, the queue to which the PCF commands are sent is always called the SYSTEM.ADMIN.COMMAND.QUEUE. z/OS On z/OS, commands are sent to SYSTEM.COMMAND.INPUT, although SYSTEM.ADMIN.COMMAND.QUEUE can be an alias for it. The command server servicing this queue sends the replies to the queue defined by the *ReplyToQ* and *ReplyToQMgr* fields in the message descriptor of the command message.

How to issue PCF command messages

Use the normal Message Queue Interface (MQI) calls, MQPUT, MQGET, and so on, to put and retrieve PCF command and response messages to and from their queues.

Note:

Ensure that the command server is running on the target queue manager for the PCF command to process on that queue manager.

For a list of supplied header files, see IBM MQ COPY, header, include and module files.

Message descriptor for a PCF command

The IBM MQ message descriptor is fully documented in MQMD - Message descriptor.

A PCF command message contains the following fields in the message descriptor:

Report
: Any valid value, as required.

MsgType
: This field must be MQMT_REQUEST to indicate a message requiring a response.

Expiry
: Any valid value, as required.

Feedback
: Set to MQFB_NONE

Encoding
: If you are sending to ▶ IBM i IBM i, Windows, UNIX or Linux systems, set this field to the encoding used for the message data; conversion is performed if necessary.

CodedCharSetId
: If you are sending to

  ▶ IBM i IBM i, Windows, UNIX or Linux systems, set this field to the coded character-set identifier used for the message data; conversion is performed if necessary.

Format
: Set to MQFMT_ADMIN.

Priority
: Any valid value, as required.

Persistence
: Any valid value, as required.

MsgId
: The sending application can specify any value, or MQMI_NONE can be specified to request the queue manager to generate a unique message identifier.

CorrelId
: The sending application can specify any value, or MQCI_NONE can be specified to indicate no correlation identifier.

ReplyToQ
: The name of the queue to receive the response.

ReplyToQMgr
The name of the queue manager for the response (or blank).

Message context fields
These fields can be set to any valid values, as required. Normally the Put message option MQPMO_DEFAULT_CONTEXT is used to set the message context fields to the default values.

If you are using a version-2 MQMD structure, you must set the following additional fields:

GroupId
Set to MQGI_NONE

MsgSeqNumber
Set to 1

Offset
Set to 0

MsgFlags
Set to MQMF_NONE

OriginalLength
Set to MQOL_UNDEFINED

Sending user data

The PCF structures can also be used to send user-defined message data. In this case the message descriptor *Format* field must be set to MQFMT_PCF.

Sending and receiving PCF messages in a specified queue:

Sending PCF messages to a specified queue

To send a message to a specified queue, the mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue. The contents of the bag are left unchanged after the call.

As input to this call, you must supply:
- An MQI connection handle.
- An object handle for the queue on which the message is to be placed.
- A message descriptor. For more information about the message descriptor, see MQMD - Message descriptor.
- Put Message Options using the MQPMO structure. For more information about the MQPMO structure, see MQPMO - Put-message options.
- The handle of the bag to be converted to a message.

  Note: If the bag contains an administration message and the mqAddInquiry call was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command recognized by the MQAI.

For a full description of the mqPutBag call, see mqPutBag.

Receiving PCF messages from a specified queue

To receive a message from a specified queue, the mqGetBag call gets a PCF message from a specified queue and converts the message data into a data bag.

As input to this call, you must supply:

- An MQI connection handle.
- An object handle of the queue from which the message is to be read.
- A message descriptor. Within the MQMD structure, the `Format` parameter must be MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF.

  Note: If the message is received within a unit of work (that is, with the MQGMO_SYNCPOINT option) and the message has an unsupported format, the unit of work can be backed out. The message is then reinstated on the queue and can be retrieved using the MQGET call instead of the mqGetBag call. For more information about the message descriptor, see MQGMO - Get-message options.
- Get Message Options using the MQGMO structure. For more information about the MQGMO structure, see MQMD - Message Descriptor.
- The handle of the bag to contain the converted message.

For a full description of the mqGetBag call, see mqGetBag.

Responses:

In response to each command, the command server generates one or more response messages. A response message has a similar format to a command message.

The PCF header has the same command identifier value as the command to which it is a response (see MQCFH - PCF header for details). The message identifier and correlation identifier are set according to the report options of the request.

If the PCF header type of the command message is MQCFT_COMMAND, standard responses only are generated. Such commands are supported on all platforms except z/OS. Older applications do not support PCF on z/OS ; the IBM MQ Windows Explorer is one such application (however, the Version 6.0 or later IBM MQ Explorer does support PCF on z/OS).

If the PCF header type of the command message is MQCFT_COMMAND_XR, either extended or standard responses are generated. Such commands are supported on z/OS and some other platforms. Commands issued on z/OS generate only extended responses. On other platforms, either type of response might be generated.

If a single command specifies a generic object name, a separate response is returned in its own message for each matching object. For response generation, a single command with a generic name is treated as multiple individual commands (except for the control field MQCFC_LAST or MQCFC_NOT_LAST). Otherwise, one command message generates one response message.

Certain PCF responses might return a structure even when it is not requested. This structure is shown in the definition of the response (Definitions of the Programmable Command Formats) as *always returned*. The reason that, for these responses, it is necessary to name the objects in the response to identify which object the data applies.

Message descriptor for a response

A response message has the following fields in the message descriptor:

MsgType
: This field is MQMT_REPLY.

MsgId
: This field is generated by the queue manager.

CorrelId
: This field is generated according to the report options of the command message.

Format
This field is MQFMT_ADMIN.

Encoding
Set to MQENC_NATIVE.

CodedCharSetId
Set to MQCCSI_Q_MGR.

Persistence
The same as in the command message.

Priority
The same as in the command message.

The response is generated with MQPMO_PASS_IDENTITY_CONTEXT.

Standard responses:

Command messages with a header type of MQCFT_COMMAND, standard responses are generated. Such commands are supported on all platforms except z/OS.

There are three types of standard response:
- OK response
- Error response
- Data response

OK response

This response consists of a message starting with a command format header, with a *CompCode* field of MQCC_OK or MQCC_WARNING.

For MQCC_OK, the *Reason* is MQRC_NONE.

For MQCC_WARNING, the *Reason* identifies the nature of the warning. In this case the command format header might be followed by one or more warning parameter structures appropriate to this reason code.

In either case, for an inquire command further parameter structures might follow as described in the following sections.

Error response

If the command has an error, one or more error response messages are sent (more than one might be sent even for a command that would normally have only a single response message). These error response messages have MQCFC_LAST or MQCFC_NOT_LAST set as appropriate.

Each such message starts with a response format header, with a *CompCode* value of MQCC_FAILED and a *Reason* field that identifies the particular error. In general, each message describes a different error. In addition, each message has either zero or one (never more than one) error parameter structures following the header. This parameter structure, if there is one, is an MQCFIN structure, with a *Parameter* field containing one of the following:
- MQIACF_PARAMETER_ID

  The *Value* field in the structure is the parameter identifier of the parameter that was in error (for example, MQCA_Q_NAME).
- MQIACF_ERROR_ID

This value is used with a *Reason* value (in the command format header) of MQRC_UNEXPECTED_ERROR. The *Value* field in the MQCFIN structure is the unexpected reason code received by the command server.

- MQIACF_SELECTOR

  This value occurs if a list structure (MQCFIL) sent with the command contains a duplicate selector or one that is not valid. The *Reason* field in the command format header identifies the error, and the *Value* field in the MQCFIN structure is the parameter value in the MQCFIL structure of the command that was in error.
- MQIACF_ERROR_OFFSET

  This value occurs when there is a data compare error on the Ping Channel command. The *Value* field in the structure is the offset of the Ping Channel compare error.
- MQIA_CODED_CHAR_SET_ID

  This value occurs when the coded character-set identifier in the message descriptor of the incoming PCF command message does not match that of the target queue manager. The *Value* field in the structure is the coded character-set identifier of the queue manager.

The last (or only) error response message is a summary response, with a *CompCode* field of MQCC_FAILED, and a *Reason* field of MQRCCF_COMMAND_FAILED. This message has no parameter structure following the header.

Data response

This response consists of an OK response (as described earlier) to an inquire command. The OK response is followed by additional structures containing the requested data as described in Definitions of the Programmable Command Formats.

Applications must not depend upon these additional parameter structures being returned in any particular order.

Extended responses:

Commands issued on z/OS generate extended responses.

There are three types of extended response:
- Message response, with type MQCFT_XR_MSG
- Item response, with type MQCFT_XR_ITEM
- Summary response, with type MQCFT_XR_SUMMARY

Each command can generate one, or more, sets of responses. Each set of responses comprises one or more messages, numbered sequentially from 1 in the *MsgSeqNumber* field of the PCF header. The *Control* field of the last (or only) response in each set has the value MQCFC_LAST. For all other responses in the set, this value is MQCFC_NOT_LAST.

Any response can include one, or more, optional MQCFBS structures in which the *Parameter* field is set to MQBACF_RESPONSE_SET, the value being a response set identifier. Identifiers are unique and identify the set of responses which contain the response. For every set of responses, there is an MQCFBS structure that identifies it.

Extended responses have at least two parameter structures:
- An MQCFBS structure with the *Parameter* field set to MQBACF_RESPONSE_ID. The value in this field is the identifier of the set of responses to which the response belongs. The identifier in the first set is arbitrary. In subsequent sets, the identifier is one previously notified in an MQBACF_RESPONSE_SET structure.

- An MQCFST structure with the *Parameter* field set to MQCACF_RESPONSE_Q_MGR_NAME, the value being the name of the queue manager from which the set of responses come.

Many responses have additional parameter structures, and these structures are described in the following sections.

You cannot determine in advance how many responses there are in a set other than by getting responses until one with MQCFC_LAST is found. Neither can you determine in advance how many sets of responses there are as any set might include MQBACF_RESPONSE_SET structures to indicate that additional sets are generated.

Extended responses to Inquire commands

Inquire commands normally generate an item response (type MQCFT_XR_ITEM) for each item found that matches the specified search criteria. The item response has a *CompCode* field in the header with a value of MQCC_OK, and a *Reason* field with a value of MQRC_NONE. It also includes other parameter structures describing the item and its requested attributes, as described in Definitions of the Programmable Command Formats.

If an item is in error, the *CompCode* field in the header has a value of MQCC_FAILED and the *Reason* field identifies the particular error. Additional parameter structures are included to identify the item.

Certain Inquire commands might return general (not name-specific) message responses in addition to the item responses. These responses are informational, or error, responses of the type MQCFT_XR_MSG.

If the Inquire command succeeds, there might, optionally, be a summary response (type MQCFT_XR_SUMMARY), with a *CompCode* value of MQCC_OK, and a *Reason* field value of MQRC_NONE.

If the Inquire command fails, item responses might be returned, and there might optionally be a summary response (type MQCFT_XR_SUMMARY), with a *CompCode* value of MQCC_FAILED, and a *Reason* field value of MQRCCF_COMMAND_FAILED.

Extended responses to commands other than Inquire

Successful commands generate message responses in which the *CompCode* field in the header has a value of MQCC_OK, and the *Reason* field has a value of MQRC_NONE. There is always at least one message; it might be informational (MQCFT_XR_MSG) or a summary (MQCFT_XR_SUMMARY). There might optionally be additional informational (type MQCFT_XR_MSG) messages. Each informational message might include a number of additional parameter structures with information about the command; see the individual command descriptions for the structures that can occur.

Commands that fail generate error message responses (type MQCFT_XR_MSG), in which the *CompCode* field in the header has a value of MQCC_FAILED and the *Reason* field identifies the particular error. Each message might include a number of additional parameter structures with information about the error: see the individual error descriptions for the structures that can occur. Informational message responses might be generated. There might, optionally, be a summary response (MQCFT_XR_SUMMARY), with a *CompCode* value of MQCC_FAILED, and a *Reason* field value of MQRCCF_COMMAND_FAILED.

Extended responses to commands using CommandScope

If a command uses the *CommandScope* parameter, or causes a command using the *CommandScope* parameter to be generated, there is an initial response set from the queue manager where the command was received. Then a separate set, or sets, of responses is generated for each queue manager to which the command is directed (as if multiple individual commands were issued). Finally, there is a response set from the receiving queue manager which includes an overall summary response (type MQCFT_XR_SUMMARY). The MQCACF_RESPONSE_Q_MGR_NAME parameter structure identifies the queue manager that generates each set.

The initial response set has the following additional parameter structures:
- MQIACF_COMMAND_INFO (MQCFIN). Possible values in this structure are MQCMDI_CMDSCOPE_ACCEPTED or MQCMDI_CMDSCOPE_GENERATED.
- MQIACF_CMDSCOPE_Q_MGR_COUNT (MQCFIN). This structure indicates the number of queue managers to which the command is sent.

Authority checking for PCF commands:

When a PCF command is processed, the `UserIdentifier` from the message descriptor in the command message is used for the required IBM MQ object authority checks. Authority checking is implemented differently on each platform as described in this topic.

The checks are performed on the system on which the command is being processed; therefore this user ID must exist on the target system and have the required authorities to process the command. If the message has come from a remote system, one way of achieving the ID existing on the target system is to have a matching user ID on both the local and remote systems.

IBM i

IBM MQ for IBM i

In order to process any PCF command, the user ID must have *dsp* authority for the IBM MQ object on the target system.

In addition, IBM MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 16.

In most cases these checks are the same checks as those checks performed by the equivalent IBM MQ CL commands issued on a local system. See the Setting up security on IBM i , for more information about the mapping from IBM MQ authorities to IBM i system authorities, and the authority requirements for the IBM MQ CL commands. Details of security concerning exits are given in the Link level security using a security exit documentation.

To process any of the following commands the user ID must be a member of the group profile QMQMADM:
- Ping Channel
- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Reset Channel
- Resolve Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener

Windows UNIX Linux

IBM MQ for Windows, UNIX and Linux systems

In order to process any PCF command, the user ID must have *dsp* authority for the queue manager object on the target system. In addition, IBM MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 16.

To process any of the following commands the user ID must belong to group *mqm*.

Note: For Windows **only**, the user ID can belong to group *Administrators* or group *mqm*.

- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Ping Channel
- Reset Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener
- Resolve Channel
- Reset Cluster
- Refresh Cluster
- Suspend Queue Manager
- Resume Queue Manager

IBM MQ for HP Integrity NonStop Server

In order to process any PCF command, the user ID must have *dsp* authority for the queue manager object on the target system. In addition, IBM MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 16.

To process any of the following commands the user ID must belong to group *mqm*:

- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Ping Channel
- Reset Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener
- Resolve Channel
- Reset Cluster
- Refresh Cluster
- Suspend Queue Manager
- Resume Queue Manager

IBM MQ Object authorities

IBM i IBM i, UNIX and Linux systems - object authorities

Table 1. Windows, HP Integrity NonStop Server,

Command	**IBM MQ object authority**	**Class authority (for object type)**
Change Authentication Information	dsp and chg	n/a
Change Channel	dsp and chg	n/a
Change Channel Listener	dsp and chg	n/a
Change Client Connection Channel	dsp and chg	n/a
Change Namelist	dsp and chg	n/a
Change Process	dsp and chg	n/a
Change Queue	dsp and chg	n/a
Change Queue Manager	chg *see Note 3 and Note 5*	n/a
Change Service	dsp and chg	n/a
Clear Queue	clr	n/a
Copy Authentication Information	dsp	crt
Copy Authentication Information (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Channel	dsp	crt
Copy Channel (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Channel Listener	dsp	crt
Copy Channel Listener (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Client Connection Channel	dsp	crt
Copy Client Connection Channel (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Namelist	dsp	crt
Copy Namelist (Replace) *see Note 1*	*from:* dsp *to:* dsp and chg	crt
Copy Process	dsp	crt
Copy Process (Replace) *see Note 1*	*from:* dsp *to:* chg	crt
Copy Queue	dsp	crt
Copy Queue (Replace) *see Note 1*	*from:* dsp *to:* dsp and chg	crt
Create Authentication Information	*(system default authentication information)* dsp	crt
Create Authentication Information (Replace) *see Note 1*	*(system default authentication information)* dsp *to:* chg	crt
Create Channel	*(system default channel)* dsp	crt
Create Channel (Replace) *see Note 1*	*(system default channel)* dsp *to:* chg	crt
Create Channel Listener	*(system default listener)* dsp	crt
Create Channel Listener (Replace) *see Note 1*	*(system default listener)* dsp *to:* chg	crt
Create Client Connection Channel	*(system default channel)* dsp	crt
Create Client Connection Channel (Replace) *see Note 1*	*(system default channel)* dsp *to:* chg	crt

Table 1. Windows, HP Integrity NonStop Server, (continued)

Command	**IBM MQ object authority**	**Class authority (for object type)**
Create Namelist	*(system default namelist)* dsp	crt
Create Namelist (Replace) *see Note 1*	*(system default namelist)* dsp *to:* dsp and chg	crt
Create Process	*(system default process)* dsp	crt
Create Process (Replace) *see Note 1*	*(system default process)* dsp *to:* chg	crt
Create Queue	*(system default queue)* dsp	crt
Create Queue (Replace) *see Note 1*	*(system default queue)* dsp *to:* dsp and chg	crt
Create Service	*(system default queue)* dsp	crt
Create Service (Replace) *see Note 1*	*(system default queue)* dsp *to:* chg	crt
Delete Authentication Information	dsp and dlt	n/a
Delete Authority Record	*(queue manager object)* chg *see Note 4*	*see Note 4*
Delete Channel	dsp and dlt	n/a
Delete Channel Listener	dsp and dlt	n/a
Delete Client Connection Channel	dsp and dlt	n/a
Delete Namelist	dsp and dlt	n/a
Delete Process	dsp and dlt	n/a
Delete Queue	dsp and dlt	n/a
Delete Service	dsp and dlt	n/a
Inquire Authentication Information	dsp	n/a
Inquire Authority Records	*see Note 4*	*see Note 4*
Inquire Channel	dsp	n/a
Inquire Channel Listener	dsp	n/a
Inquire Channel Status (for **ChannelType** MQCHT_CLSSDR)	inq	n/a
Inquire Client Connection Channel	dsp	n/a
Inquire Namelist	dsp	n/a
Inquire Process	dsp	n/a
Inquire Queue	dsp	n/a
Inquire Queue Manager	*see note 3*	n/a
Inquire Queue Status	dsp	n/a
Inquire Service	dsp	n/a
Ping Channel	ctrl	n/a
Ping Queue Manager	*see note 3*	n/a
Refresh Queue Manager	(queue manager object) chg	n/a
Refresh Security (for **SecurityType** MQSECTYPE_SSL)	(queue manager object) chg	n/a
Reset Channel	ctrlx	n/a
Reset Queue Manager	(queue manager object) chg	n/a
Reset Queue Statistics	dsp and chg	n/a
Resolve Channel	ctrlx	n/a
Set Authority Record	*(queue manager object)* chg *see Note 4*	*see Note 4*

Table 1. Windows, HP Integrity NonStop Server, (continued)

Command	**IBM MQ object authority**	**Class authority (for object type)**
Start Channel	ctrl	n/a
Stop Channel	ctrl	n/a
Stop Connection	(queue manager object) chg	n/a
Start Listener	ctrl	n/a
Stop Listener	ctrl	n/a
Start Service	ctrl	n/a
Stop Service	ctrl	n/a
Escape	*see Note 2*	*see Note 2*

Notes:

1. This command applies if the object to be replaced does exist, otherwise the authority check is as for Create, or Copy without Replace.
2. The required authority is determined by the MQSC command defined by the escape text, and it is equivalent to one of the previous commands.
3. In order to process any PCF command, the user ID must have dsp authority for the queue manager object on the target system.
4. This PCF command is authorized unless the command server has been started with the -a parameter. By default the command server starts when the Queue Manager is started, and without the -a parameter. See the System Administration Guide for further information.
5. Granting a user ID *chg* authority for a queue manager gives the ability to set authority records for all groups and users. Do not grant this authority to ordinary users or applications.

IBM MQ also supplies some channel security exit points so that you can supply your own user exit programs for security checking. Details are given in Displaying a channel manual.

z/OS

IBM MQ for z/OS

See Task 1: Identify the z/OS system parameters for information about authority checking on z/OS.

Using the MQAI to simplify the use of PCFs

The MQAI is an administration interface to IBM MQ that is available on the AIX, HP-UX, IBM i, Linux, Solaris, and Windows platforms.

The MQAI performs administration tasks on a queue manager through the use of *data bags*. Data bags allow you to handle properties (or parameters) of objects in a way that is easier than using PCFs.

The advantages of using the MQAI are as follows:

Simplify the use of PCF messages

The MQAI is an easier way to administer IBM MQ. If you use the MQAI, you do not have to write your own PCF messages. This avoids the problems associated with complex data structures.

To pass parameters in programs written using MQI calls, the PCF message must contain the command, and details of the string or integer data. To create this configuration manually, you have to add several statements in your program for every structure, and you have to allocate memory space. This task can be long and laborious.

Programs written using the MQAI pass parameters into the appropriate data bag, and you need only one statement for each structure. The use of the MQAI data bags removes the need for you to handle arrays and allocate storage, and provides some degree of isolation from the details of the PCF.

Handle error conditions more easily
It is difficult to get return codes back from PCF commands. The MQAI makes it easier for the program to handle error conditions.

After you have created and populated your data bag, you can send an administration command message to the command server of a queue manager, using the **mqExecute** call. This call waits for any response messages. The **mqExecute** call handles the exchange with the command server, and returns responses in a *response bag*.

For more information about the MQAI, see "Introduction to the IBM MQ Administration Interface (MQAI)."

Introduction to the IBM MQ Administration Interface (MQAI)

IBM MQ Administration Interface (MQAI) is a programming interface to IBM MQ. It performs administration tasks on an IBM MQ queue manager using data bags to handle properties (or parameters) of objects in a way that is easier than using Programmable Command Formats (PCFs).

MQAI concepts and terminology

The MQAI is a programming interface to IBM MQ, using the C language and also Visual Basic for Windows. It is available on platforms other than z/OS.

It performs administration tasks on an IBM MQ queue manager using data bags. Data bags allow you to handle properties (or parameters) of objects in a way that is easier than using the other administration interface, Programmable Command Formats (PCFs). The MQAI offers easier manipulation of PCFs than using the MQGET and MQPUT calls.

For more information about data bags, see "Data bags" on page 46. For more information about PCFs, see "Introduction to Programmable Command Formats" on page 6

Use of the MQAI

You can use the MQAI to:

- Simplify the use of PCF messages. The MQAI is an easy way to administer IBM MQ; you do not have to write your own PCF messages and thus avoid the problems associated with complex data structures.
- Handle error conditions more easily. It is difficult to get return codes back from the IBM MQ script (MQSC) commands, but the MQAI makes it easier for the program to handle error conditions.
- Exchange data between applications. The application data is sent in PCF format and packed and unpacked by the MQAI. If your message data consists of integers and character strings, you can use the MQAI to take advantage of IBM MQ built-in data conversion for PCF data. This avoids the need to write data-conversion exits. For more information on using MQAI to administer IBM MQ and to exchange data between applications, see "Using the MQAI to simplify the use of PCFs" on page 18.

Examples of using the MQAI

The list shown gives some example programs that demonstrate the use of MQAI. The samples perform the following tasks:

1. Create a local queue. "Creating a local queue (amqsaicq.c)" on page 20

2. Display events on the screen using a simple event monitor. “Displaying events using an event monitor (amqsaiem.c)” on page 25
3. Print a list of all local queues and their current depths. “Inquiring about queues and printing information (amqsailq.c)” on page 39
4. Print a list of all channels and their types. “Inquire channel objects (amqsaicl.c)” on page 33

Building your MQAI application

To build your application using the MQAI, you link to the same libraries as you do for IBM MQ. For information on how to build your IBM MQ applications, see Building a procedural application.

Hints and tips for configuring IBM MQ using MQAI

The MQAI uses PCF messages to send administration commands to the command server rather than dealing directly with the command server itself. Tips for configuring IBM MQ using the MQAI can be found in “Hints and tips for configuring IBM MQ” on page 44

IBM MQ Administration Interface (MQAI)

IBM MQ for Windows, AIX, IBM i IBM i, Linux, HP-UX, and Solaris support the IBM MQ Administration Interface (MQAI). The MQAI is a programming interface to IBM MQ that gives you an alternative to the MQI, for sending and receiving PCFs.

The MQAI uses *data bags* which allow you to handle properties (or parameters) of objects more easily than using PCFs directly by way of the MQAI.

The MQAI provides easier programming access to PCF messages by passing parameters into the data bag, so that only one statement is required for each structure. This access removes the need for the programmer to handle arrays and allocate storage, and provides some isolation from the details of PCF.

The MQAI administers IBM MQ by sending PCF messages to the command server and waiting for a response.

The MQAI is described in the second section of this manual. See the Using Java™ documentation for a description of a component object model interface to the MQAI.

Creating a local queue (amqsaicq.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAICQ.C                                                   */
/*                                                                            */
/* Description:  Sample C program to create a local queue using the          */
/*               IBM MQ Administration Interface (MQAI).                     */
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                        */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999, 2005                           */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAICQ is a sample C program that creates a local queue and is an     */
/*    example of the use of the mqExecute call.                               */
/*                                                                            */
/*     - The name of the queue to be created is a parameter to the program.   */
/*                                                                            */
/*     - A PCF command is built by placing items into an MQAI bag.            */
/*       These are:-                                                          */
/*            - The name of the queue                                         */
/*            - The type of queue required, which, in this case, is local.    */
/*                                                                            */
/*     - The mqExecute call is executed with the command MQCMD_CREATE_Q.      */
/*       The call generates the correct PCF structure.                        */
/*       The call receives the reply from the command server and formats into */
/*       the response bag.                                                    */
/*                                                                            */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server then the code returned by the   */
/*       command server is retrieved from the system bag that is              */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                            */
/* Note: The command server must be running.                                  */
/*                                                                            */
/*                                                                            */

/******************************************************************************/
/*                                                                            */
/* AMQSAICQ has 2 parameters - the name of the local queue to be created      */
/*                           - the queue manager name (optional)              */
/*                                                                            */
/******************************************************************************/
/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                         /* MQI                      */
#include <cmqcfc.h>                       /* PCF                      */
#include <cmqbc.h>                        /* MQAI                     */

void CheckCallResult(MQCHAR *, MQLONG , MQLONG );
void CreateLocalQueue(MQHCONN, MQCHAR *);

int main(int argc, char *argv[])
{
   MQHCONN hConn;                         /* handle to IBM MQ connection   */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name          */
   MQLONG connReason;                     /* MQCONN reason code            */
   MQLONG compCode;                       /* completion code               */
   MQLONG reason;                         /* reason code                   */

   /***************************************************************************/
   /* First check the required parameters                                     */
   /***************************************************************************/
   printf("Sample Program to Create a Local Queue\n");
   if (argc < 2)
   {
     printf("Required parameter missing - local queue name\n");
     exit(99);
   }

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 2)
      strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
      MQCONN(QMName, &hConn, &compCode, &connReason);
```

```
/******************************************************************************/
/* Report reason and stop if connection failed                                */
/******************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }


/******************************************************************************/
/* Call the routine to create a local queue, passing the handle to the       */
/* queue manager and also passing the name of the queue to be created.       */
/******************************************************************************/
   CreateLocalQueue(hConn, argv[1]);

   /***************************************************************************/
   /* Disconnect from the queue manager if not already connected             */
   /***************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }
   return 0;

}
/******************************************************************************/
/*                                                                            */
/* Function:    CreateLocalQueue                                              */
/* Description: Create a local queue by sending a PCF command to the command  */
/*              server.                                                       */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Handle to the queue manager                             */
/*                    Name of the queue to be created                         */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: The mqExecute call is executed with the command MQCMD_CREATE_Q.     */
/*        The call generates the correct PCF structure.                       */
/*        The default options to the call are used so that the command is sent*/
/*        to the SYSTEM.ADMIN.COMMAND.QUEUE.                                  */
/*        The reply from the command server is placed on a temporary dynamic  */
/*        queue.                                                              */
/*        The reply is read from the temporary queue and formatted into the   */
/*        response bag.                                                       */
/*                                                                            */
/*        The completion code from the mqExecute call is checked and if there */
/*        is a failure from the command server then the code returned by the  */
/*        command server is retrieved from the system bag that is             */
/*        embedded in the response bag to the mqExecute call.                 */
/*                                                                            */
/******************************************************************************/
void CreateLocalQueue(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG reason;                          /* reason code                    */
   MQLONG compCode;                        /* completion code                */
   MQHBAG commandBag = MQHB_UNUSABLE_HBAG; /* command bag for mqExecute      */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute     */
   MQHBAG resultBag;                       /* result bag from mqExecute      */
   MQLONG mqExecuteCC;                     /* mqExecute completion code      */
   MQLONG mqExecuteRC;                     /* mqExecute reason code          */

   printf("\nCreating Local Queue %s\n\n", qName);
```

```
/***************************************************************************/
/* Create a command Bag for the mqExecute call. Exit the function if the   */
/* create fails.                                                           */
/***************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &commandBag, &compCode, &reason);
CheckCallResult("Create the command bag", compCode, reason);
if (compCode !=MQCC_OK)
   return;

/***************************************************************************/
/* Create a response Bag for the mqExecute call, exit the function if the  */
/* create fails.                                                           */
/***************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
CheckCallResult("Create the response bag", compCode, reason);
if (compCode !=MQCC_OK)
   return;

/***************************************************************************/
/* Put the name of the queue to be created into the command bag. This will */
/* be used by the mqExecute call.                                          */
/***************************************************************************/
mqAddString(commandBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, qName, &compCode,
            &reason);
CheckCallResult("Add q name to command bag", compCode, reason);

/***************************************************************************/
/* Put queue type of local into the command bag. This will be used by the  */
/* mqExecute call.                                                         */
/***************************************************************************/
mqAddInteger(commandBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
CheckCallResult("Add q type to command bag", compCode, reason);

/***************************************************************************/
/* Send the command to create the required local queue.                    */
/* The mqExecute call will create the PCF structure required, send it to   */
/* the command server and receive the reply from the command server into   */
/* the response bag.                                                       */
/***************************************************************************/
mqExecute(hConn,                   /* IBM MQ connection handle             */
          MQCMD_CREATE_Q,          /* Command to be executed               */
          MQHB_NONE,               /* No options bag                       */
          commandBag,              /* Handle to bag containing commands    */
          responseBag,             /* Handle to bag to receive the response*/
          MQHO_NONE,               /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,               /* Create a dynamic q for the response  */
          &compCode,           /* Completion code from the mqExecute   */
          &reason);            /* Reason code from mqExecute call      */

if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName>\n")
   MQDISC(&hConn, &compCode, &reason);
   CheckCallResult("MQDISC", compCode, reason);
   exit(98);
}


/***************************************************************************/
/* Check the result from mqExecute call and find the error if it failed.   */
/***************************************************************************/
if ( compCode == MQCC_OK )
   printf("Local queue %s successfully created\n", qName);
else
{
      printf("Creation of local queue %s failed: Completion Code = %d
               qName, compCode, reason);
      if (reason == MQRCCF_COMMAND_FAILED)
      {
         /*********************************************************************/
         /* Get the system bag handle out of the mqExecute response bag.      */
         /* This bag contains the reason from the command server why the      */
         /* command failed.                                                   */
         /*********************************************************************/
         mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &resultBag, &compCode,
                      &reason);
         CheckCallResult("Get the result bag handle", compCode, reason);

         /*********************************************************************/
         /* Get the completion code and reason code, returned by the command  */
         /* server, from the embedded error bag.                              */
         /*********************************************************************/
         mqInquireInteger(resultBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                          &compCode, &reason);
         CheckCallResult("Get the completion code from the result bag",
                          compCode, reason);
         mqInquireInteger(resultBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                          &compCode, &reason);
         CheckCallResult("Get the reason code from the result bag", compCode,
                          reason);
         printf("Error returned by the command server: Completion code = %d :
                 Reason = %d\n", mqExecuteCC, mqExecuteRC);
      }
   }
   /****************************************************************************/
   /* Delete the command bag if successfully created.                         */
   /****************************************************************************/
   if (commandBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&commandBag, &compCode, &reason);
      CheckCallResult("Delete the command bag", compCode, reason);
   }

   /****************************************************************************/
   /* Delete the response bag if successfully created.                        */
   /****************************************************************************/
   if (responseBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&responseBag, &compCode, &reason);
      CheckCallResult("Delete the response bag", compCode, reason);
   }
} /* end of CreateLocalQueue */


/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the    */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d :
                 Reason = %d\n", callText, cc, rc);

}
```

Displaying events using an event monitor (amqsaiem.c)

```
******************************************************************************/
/*                                                                            */
/* Program name: AMQSAIEM.C                                                   */
/*                                                                            */
/* Description:  Sample C program to demonstrate a basic event monitor        */
/*               using the IBM MQ Admin Interface (MQAI).                     */
/* Licensed Materials - Property of IBM                                       */
/*                                                                            */
/* 63H9336                                                                    */
/* (c) Copyright IBM Corp. 1999, 2005 All Rights Reserved.                    */
/*                                                                            */
/* US Government Users Restricted Rights - Use, duplication or                */
/* disclosure restricted by GSA ADP Schedule Contract with                    */
/* IBM Corp.                                                                  */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAIEM is a sample C program that demonstrates how to write a simple  */
/*    event monitor using the mqGetBag call and other MQAI calls.             */
/*                                                                            */
/*    The name of the event queue to be monitored is passed as a parameter    */
/*    to the program. This would usually be one of the system event queues:-  */
/*            SYSTEM.ADMIN.QMGR.EVENT        Queue Manager events             */
/*            SYSTEM.ADMIN.PERFM.EVENT       Performance events               */
/*            SYSTEM.ADMIN.CHANNEL.EVENT     Channel events                   */
/*            SYSTEM.ADMIN.LOGGER.EVENT      Logger events                    */
/*                                                                            */
/*    To monitor the queue manager event queue or the performance event queue,*/
/*    the attributes of the queue manager need to be changed to enable        */
/*    these events. For more information about this, see Part 1 of the        */
/*    Programmable System Management book. The queue manager attributes can   */
/*    be changed using either MQSC commands or the MQAI interface.            */
/*    Channel events are enabled by default.                                  */
/*                                                                            */
/* Program logic                                                              */
/*    Connect to the Queue Manager.                                           */
/*    Open the requested event queue with a wait interval of 30 seconds.      */
/*    Wait for a message, and when it arrives get the message from the queue  */
/*    and format it into an MQAI bag using the mqGetBag call.                 */
/*    There are many types of event messages and it is beyond the scope of    */
/*    this sample to program for all event messages. Instead the program      */
/*    prints out the contents of the formatted bag.                           */
/*    Loop around to wait for another message until either there is an error  */
/*    or the wait interval of 30 seconds is reached.                          */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAIEM has 2 parameters - the name of the event queue to be monitored    */
/*                            - the queue manager name (optional)             */
/*                                                                            */
/*****************************************************************************

/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
```

```
#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */

/******************************************************************************/
/* Macros                                                                     */
/******************************************************************************/
#if MQAT_DEFAULT == MQAT_WINDOWS_NT
  #define Int64 "I64"
#elif defined(MQ_64_BIT)
  #define Int64 "l"
#else
  #define Int64 "ll"
#endif

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);
void GetQEvents(MQHCONN, MQCHAR *);
int PrintBag(MQHBAG);
int PrintBagContents(MQHBAG, int);

/******************************************************************************/
/* Function: main                                                             */
/******************************************************************************/
int main(int argc, char *argv[])
{
   MQHCONN hConn;                          /* handle to connection            */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QM name               */
   MQLONG reason;                          /* reason code                     */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */

   /***************************************************************************/
   /* First check the required parameters                                     */
   /***************************************************************************/
   printf("Sample Event Monitor (times out after 30 secs)\n");
   if (argc < 2)
   {
     printf("Required parameter missing - event queue to be monitored\n");
     exit(99);
   }

   /**************************************************************************/
   /* Connect to the queue manager                                           */
   /**************************************************************************/
   if (argc > 2)
     strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(QMName, &hConn, &compCode, &connReason);
   /***************************************************************************/
   /* Report the reason and stop if the connection failed                     */
   /***************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }

   /***************************************************************************/
   /* Call the routine to open the event queue and format any event messages  */
   /* read from the queue.                                                    */
   /***************************************************************************/
   GetQEvents(hConn, argv[1]);

   /***************************************************************************/
   /* Disconnect from the queue manager if not already connected              */
   /***************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }

   return 0;

}

/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the    */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d : Reason = %d\n",
                 callText, cc, rc);

}

/******************************************************************************/
/*                                                                            */
/* Function: GetQEvents                                                       */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Handle to the queue manager                             */
/*                    Name of the event queue to be monitored                 */
/*                                                                            */
/* Output Parameters: None                                                    */
/*
/* Logic:   Open the event queue.                                             */
/*          Get a message off the event queue and format the message into     */
/*          a bag.                                                            */
/*          A real event monitor would need to be programmed to deal with     */
/*          each type of event that it receives from the queue. This is       */
/*          outside the scope of this sample, so instead, the contents of     */
/*          the bag are printed.                                              */
/*          The program waits for 30 seconds for an event message and then    */
/*          terminates if no more messages are available.                     */
/*                                                                            */
/******************************************************************************/
void GetQEvents(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG openReason;                       /* MQOPEN reason code            */
   MQLONG reason;                           /* reason code                   */
   MQLONG compCode;                         /* completion code               */
   MQHOBJ eventQueue;                       /* handle to event queue         */

   MQHBAG eventBag = MQHB_UNUSABLE_HBAG;    /* event bag to receive event msg */
   MQOD   od = {MQOD_DEFAULT};              /* Object Descriptor             */
   MQMD   md = {MQMD_DEFAULT};              /* Message Descriptor            */
```

```
MQGMO  gmo = {MQGMO_DEFAULT};            /* get message options           */
MQLONG bQueueOK = 1;                     /* keep reading msgs while true  */

/**************************************************************************/
/* Create an Event Bag in which to receive the event.                     */
/* Exit the function if the create fails.                                 */
/**************************************************************************/
mqCreateBag(MQCBO_USER_BAG, &eventBag, &compCode, &reason);
CheckCallResult("Create event bag", compCode, reason);
if (compCode !=MQCC_OK)
   return;

/**************************************************************************/
/* Open the event queue chosen by the user                                */
/**************************************************************************/
strncpy(od.ObjectName, qName, (size_t)MQ_Q_NAME_LENGTH);
MQOPEN(hConn, &od, MQOO_INPUT_AS_Q_DEF+MQOO_FAIL_IF_QUIESCING, &eventQueue,
       &compCode, &openReason);
CheckCallResult("Open event queue", compCode, openReason);

/**************************************************************************/
/* Set the GMO options to control the action of the get message from the  */
/* queue.                                                                 */
/**************************************************************************/
gmo.WaitInterval = 30000;               /* 30 second wait for message     */
gmo.Options = MQGMO_WAIT + MQGMO_FAIL_IF_QUIESCING + MQGMO_CONVERT;
gmo.Version = MQGMO_VERSION_2;          /* Avoid need to reset Message ID  */
gmo.MatchOptions = MQMO_NONE;           /* and Correlation ID after every  */
                                 /* mqGetBag
/**************************************************************************/
/* If open fails, we cannot access the queue and must stop the monitor.   */
/**************************************************************************/
if (compCode != MQCC_OK)
   bQueueOK = 0;

/**************************************************************************/
/* Main loop to get an event message when it arrives                      */
/**************************************************************************/
while (bQueueOK)
{
  printf("\nWaiting for an event\n");

  /************************************************************************/
  /* Get the message from the event queue and convert it into the event   */
  /* bag.                                                                 */
  /************************************************************************/
  mqGetBag(hConn, eventQueue, &md, &gmo, eventBag, &compCode, &reason);

  /************************************************************************/
  /* If get fails, we cannot access the queue and must stop the monitor.  */
  /************************************************************************/
  if (compCode != MQCC_OK)
  {
     bQueueOK = 0;

     /*********************************************************************/
     /* If get fails because no message available then we have timed out, */
     /* so report this, otherwise report an error.                        */
     /*********************************************************************/
     if (reason == MQRC_NO_MSG_AVAILABLE)
     {
        printf("No more messages\n");
     }
     else
     {
        CheckCallResult("Get bag", compCode, reason);
     }
```

```
      }

      /************************************************************************/
      /* Event message read - Print the contents of the event bag             */
      /************************************************************************/
      else
      {
        if ( PrintBag(eventBag) )
            printf("\nError found while printing bag contents\n");

      }  /* end of msg found */
    } /* end of main loop */
    /**************************************************************************/
    /* Close the event queue if successfully opened                           */
    /**************************************************************************/
    if (openReason == MQRC_NONE)
    {
       MQCLOSE(hConn, &eventQueue, MQCO_NONE, &compCode, &reason);
       CheckCallResult("Close event queue", compCode, reason);
    }

    /**************************************************************************/
    /* Delete the event bag if successfully created.                          */
    /**************************************************************************/
    if (eventBag != MQHB_UNUSABLE_HBAG)
    {
       mqDeleteBag(&eventBag, &compCode, &reason);
       CheckCallResult("Delete the event bag", compCode, reason);
    }

} /* end of GetQEvents */

/******************************************************************************/
/*                                                                            */
/* Function: PrintBag                                                         */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Bag Handle                                              */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Returns:           Number of errors found                                  */
/*                                                                            */
/* Logic: Calls PrintBagContents to display the contents of the bag.          */
/*                                                                            */
/*****************************************************************************

int PrintBag(MQHBAG dataBag)
{
    int errors;

    printf("\n");
    errors = PrintBagContents(dataBag, 0);
    printf("\n");

    return errors;
}

/******************************************************************************/
/*                                                                            */
/* Function: PrintBagContents                                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Bag Handle                                              */
/*                    Indentation level of bag                                */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Returns:           Number of errors found                                  */
/*                                                                            */
/* Logic: Count the number of items in the bag                                */
/*        Obtain selector and item type for each item in the bag.             */
/*        Obtain the value of the item depending on item type and display the */
/*        index of the item, the selector and the value.                      */
/*        If the item is an embedded bag handle then call this function again */
/*        to print the contents of the embedded bag increasing the            */
/*        indentation level.                                                  */
/*                                                                            */
/******************************************************************************/
int PrintBagContents(MQHBAG dataBag, int indent)
{
   /***************************************************************************/
   /* Definitions                                                             */
   /***************************************************************************/
   #define LENGTH 500                        /* Max length of string to be read*/
   #define INDENT 4                          /* Number of spaces to indent     */
                                             /* embedded bag display           */


   /***************************************************************************/
   /* Variables                                                               */
   /***************************************************************************/
   MQLONG  itemCount;                        /* Number of items in the bag     */
   MQLONG  itemType;                         /* Type of the item               */
   int     i;                                /* Index of item in the bag       */
   MQCHAR  stringVal[LENGTH+1];              /* Value if item is a string      */
   MQBYTE  byteStringVal[LENGTH];            /* Value if item is a byte string */
   MQLONG  stringLength;                     /* Length of string value         */
   MQLONG  ccsid;                            /* CCSID of string value          */
   MQINT32 iValue;                           /* Value if item is an integer    */
   MQINT64 i64Value;                         /* Value if item is a 64-bit      */
                                             /* integer                        */
   MQLONG  selector;                         /* Selector of item               */
   MQHBAG  bagHandle;                        /* Value if item is a bag handle  */
   MQLONG  reason;                           /* reason code                    */
   MQLONG  compCode;                         /* completion code                */
   MQLONG  trimLength;                       /* Length of string to be trimmed */
   int     errors = 0;                       /* Count of errors found          */
   char    blanks[] = "                        "; /* Blank string used to      */
                                                  /* indent display            */

   /***************************************************************************/
   /* Count the number of items in the bag                                    */
   /***************************************************************************/
   mqCountItems(dataBag, MQSEL_ALL_SELECTORS, &itemCount, &compCode, &reason);

   if (compCode != MQCC_OK)
      errors++;
   else
   {
      printf("
      printf("
      printf("
   }

   /***************************************************************************/
   /* If no errors found, display each item in the bag                        */
   /***************************************************************************/
   if (!errors)
   {
      for (i = 0; i < itemCount; i++)
      {
```

```
/******************************************************************/
/* First inquire the type of the item for each item in the bag      */
/******************************************************************/
mqInquireItemInfo(dataBag,           /* Bag handle               */
                  MQSEL_ANY_SELECTOR,  /* Item can have any selector*/
                  i,                  /* Index position in the bag */
                  &selector,          /* Actual value of selector  */
                                      /* returned by call          */
                  &itemType,          /* Actual type of item       */
                                      /* returned by call          */
                  &compCode,          /* Completion code           */
                  &reason);           /* Reason Code               */

if (compCode != MQCC_OK)
   errors++;

switch(itemType)
{
case MQITEM_INTEGER:
     /***************************************************************/
     /* Item is an integer. Find its value and display its index,   */
     /* selector and value.                                         */
     /***************************************************************/
     mqInquireInteger(dataBag,        /* Bag handle               */
                      MQSEL_ANY_SELECTOR, /* Allow any selector     */
                      i,              /* Index position in the bag */
                      &iValue,        /* Returned integer value
                      &compCode,      /* Completion code           */
                      &reason);       /* Reason Code               */

     if (compCode != MQCC_OK)
        errors++;
     else
        printf("%.*s  %-2d     %-4d     (%d)\n",
                indent, blanks, i, selector, iValue);
     break

case MQITEM_INTEGER64:
     /***************************************************************/
     /* Item is a 64-bit integer. Find its value and display its    */
     /* index, selector and value.                                  */
     /***************************************************************/
     mqInquireInteger64(dataBag,      /* Bag handle               */
                        MQSEL_ANY_SELECTOR, /* Allow any selector   */
                        i,            /* Index position in the bag */
                        &i64Value,    /* Returned integer value    */
                        &compCode,    /* Completion code           */
                        &reason);     /* Reason Code               */

     if (compCode != MQCC_OK)
        errors++;
     else
        printf("%.*s  %-2d     %-4d     (%"Int64"d)\n",
                indent, blanks, i, selector, i64Value);
     break;


case MQITEM_STRING:
     /***************************************************************/
     /* Item is a string. Obtain the string in a buffer, prepare    */
     /* the string for displaying and display the index, selector,  */
     /* string and Character Set ID.                                */
     /***************************************************************/
     mqInquireString(dataBag,         /* Bag handle               */
                     MQSEL_ANY_SELECTOR, /* Allow any selector      */
                     i,               /* Index position in the bag */
```

```
                         LENGTH,           /* Maximum length of buffer  */
                         stringVal,        /* Buffer to receive string  */
                         &stringLength,    /* Actual length of string   */
                         &ccsid,           /* Coded character set id    */
                         &compCode,        /* Completion code           */
                         &reason);         /* Reason Code               */

         /***************************************************************/
         /* The call can return a warning if the string is too long for */
         /* the output buffer and has been truncated, so only check     */
         /* explicitly for call failure.                                */
         /***************************************************************/
         if (compCode == MQCC_FAILED)
             errors++;
         else
         {
            /************************************************************/
            /* Remove trailing blanks from the string and terminate with*/
            /* a null. First check that the string should not have been */
            /* longer than the maximum buffer size allowed.             */
            /************************************************************/
            if (stringLength > LENGTH)
               trimLength = LENGTH;
            else
               trimLength = stringLength;
            mqTrim(trimLength, stringVal, stringVal, &compCode, &reason);
            printf("%.*s  %-2d     %-4d     '%s' %d\n",
                    indent, blanks, i, selector, stringVal, ccsid);
         }
         break;

    case MQITEM_BYTE_STRING:
         /***************************************************************/
         /* Item is a byte string. Obtain the byte string in a buffer,  */
         /* prepare the byte string for displaying and display the      */
         /* index, selector and string.                                 */
         /***************************************************************/
         mqInquireByteString(dataBag,      /* Bag handle                */
                             MQSEL_ANY_SELECTOR, /* Allow any selector  */
                             i,            /* Index position in the bag */
                             LENGTH,       /* Maximum length of buffer  */
                             byteStringVal, /* Buffer to receive string */
                             &stringLength, /* Actual length of string  */
                             &compCode,    /* Completion code           */
                             &reason);     /* Reason Code

         /***************************************************************/
         /* The call can return a warning if the string is too long for */
         /* the output buffer and has been truncated, so only check     */
         /* explicitly for call failure.                                */
         /***************************************************************/
         if (compCode == MQCC_FAILED)
             errors++;
         else
         {
            printf("%.*s  %-2d     %-4d     X'",
                   indent, blanks, i, selector);

            for (i = 0 ; i < stringLength ; i++)
               printf("

            printf("'\n");
         }
         break;

    case MQITEM_BAG:
         /***************************************************************/
```

```
               /* Item is an embedded bag handle, so call the PrintBagContents*/
               /* function again to display the contents.                      */
               /***************************************************************/
               mqInquireBag(dataBag,             /* Bag handle                */
                            MQSEL_ANY_SELECTOR,  /* Allow any selector        */
                            i,                   /* Index position in the bag */
                            &bagHandle,          /* Returned embedded bag hdle*/
                            &compCode,           /* Completion code           */
                            &reason);            /* Reason Code               */

               if (compCode != MQCC_OK)
                  errors++;
               else
               {
                  printf("%.*s  %-2d     %-4d     (%d)\n", indent, blanks, i,
                          selector, bagHandle);
                  if (selector == MQHA_BAG_HANDLE)
                     printf("
                  else
                     printf("
                  PrintBagContents(bagHandle, indent+INDENT);
               }
               break;

          default:
               printf("
          }
      }
   }
   return errors;
}
```

Inquire channel objects (amqsaicl.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAICL.C                                                   */
/*                                                                            */
/* Description:  Sample C program to inquire channel objects                  */
/*               using the IBM MQ Administration Interface (MQAI)             */
/*                                                                            */
/* <N_OCO_COPYRIGHT>                                                          */
/* Licensed Materials - Property of IBM                                       */
/*                                                                            */
/* 63H9336                                                                    */
/* (c) Copyright IBM Corp. 2008 All Rights Reserved.                          */
/*                                                                            */
/* US Government Users Restricted Rights - Use, duplication or                */
/* disclosure restricted by GSA ADP Schedule Contract with                    */
/* IBM Corp.                                                                  */
/* <NOC_COPYRIGHT>                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAICL is a sample C program that demonstrates how to inquire         */
/*    attributes of the local queue manager using the MQAI interface. In      */
/*    particular, it inquires all channels and their types.                   */
/*                                                                            */
/*     - A PCF command is built from items placed into an MQAI administration */
/*       bag.                                                                 */
/*       These are:-                                                          */
/*            - The generic channel name "*"                                  */
/*            - The attributes to be inquired. In this sample we just want    */
/*              name and type attributes                                      */
/*                                                                            */
/*     - The mqExecute MQCMD_INQUIRE_CHANNEL call is executed.                */
/*       The call generates the correct PCF structure.                        */
/*       The default options to the call are used so that the command is sent */
/*       to the SYSTEM.ADMIN.COMMAND.QUEUE.                                 */
/*       The reply from the command server is placed on a temporary dynamic   */
/*       queue.                                                             */
/*       The reply from the MQCMD_INQUIRE_CHANNEL is read from the          */
/*       temporary queue and formatted into the response bag.               */
/*                                                                          */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server, then the code returned by the  */
/*       command server is retrieved from the system bag that has been        */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                          */
/* Note: The command server must be running.                                */
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/* AMQSAICL has 2 parameter - the queue manager name (optional)             */
/*                          - output file (optional) default varies         */
/****************************************************************************/

/****************************************************************************/
/* Includes                                                                 */
/****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#if (MQAT_DEFAULT == MQAT_OS400)
#include <recio.h>
#endif

#include <cmqc.h>                          /* MQI                            */
#include <cmqcfc.h>                        /* PCF                            */
#include <cmqbc.h>                         /* MQAI                           */
#include <cmqxc.h>                         /* MQCD                           */

/****************************************************************************/
/* Function prototypes                                                      */
/****************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);

/****************************************************************************/
/* DataTypes                                                                */
/****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
typedef _RFILE OUTFILEHDL;
#else
typedef FILE OUTFILEHDL;
#endif

/****************************************************************************/
/* Constants                                                                */
/****************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
const struct
{
  char name[9];
} ChlTypeMap[9] =
{
  "*SDR     ",    /* MQCHT_SENDER    */
  "*SVR     ",    /* MQCHT_SERVER    */
  "*RCVR    ",    /* MQCHT_RECEIVER  */
  "*RQSTR   ",    /* MQCHT_REQUESTER */
  "*ALL     ",    /* MQCHT_ALL       */
  "*CLTCN   ",    /* MQCHT_CLNTCONN  */
  "*SVRCONN ",    /* MQCHT_SVRCONN   */
  "*CLUSRCVR",    /* MQCHT_CLUSRCVR  */
  "*CLUSSDR "     /* MQCHT_CLUSSDR   */
```

```
};
#else
const struct
{
  char name[9];
} ChlTypeMap[9] =
{
  "sdr      ",    /* MQCHT_SENDER    */
  "svr      ",    /* MQCHT_SERVER    */
  "rcvr     ",    /* MQCHT_RECEIVER  */
  "rqstr    ",    /* MQCHT_REQUESTER */
  "all      ",    /* MQCHT_ALL       */
  "cltconn  ",    /* MQCHT_CLNTCONN  */
  "svrcn    ",    /* MQCHT_SVRCONN   */
  "clusrcvr ",    /* MQCHT_CLUSRCVR  */
  "clussdr  "     /* MQCHT_CLUSSDR   */
};
#endif

/********************************************************************************/
/* Macros                                                                       */
/********************************************************************************/
#if (MQAT_DEFAULT == MQAT_OS400)
  #define OUTFILE "QTEMP/AMQSAICL(AMQSAICL)"
  #define OPENOUTFILE(hdl, fname) \
    (hdl) = _Ropen((fname),"wr, rtncode=Y");
  #define CLOSEOUTFILE(hdl) \
    _Rclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    _Rwrite((hdl),(buf),(buflen));

#elif (MQAT_DEFAULT == MQAT_UNIX)
  #define OUTFILE "/tmp/amqsaicl.txt"
  #define OPENOUTFILE(hdl, fname) \
    (hdl) = fopen((fname),"w");
  #define CLOSEOUTFILE(hdl) \
    fclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    fwrite((buf),(buflen),1,(hdl)); fflush((hdl));

#else
  #define OUTFILE "amqsaicl.txt"
  #define OPENOUTFILE(fname) \
    fopen((fname),"w");
  #define CLOSEOUTFILE(hdl) \
    fclose((hdl));
  #define WRITEOUTFILE(hdl, buf, buflen) \
    fwrite((buf),(buflen),1,(hdl)); fflush((hdl));

#endif

#define ChlType2String(t) ChlTypeMap[(t)-1].name

/********************************************************************************/
/* Function: main                                                               */
/********************************************************************************/
int main(int argc, char *argv[])
{
   /*****************************************************************************/
   /* MQAI variables                                                            */
   /*****************************************************************************/
   MQHCONN hConn;                          /* handle to MQ connection         */
   MQCHAR qmName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name             */
   MQLONG reason;                          /* reason code                     */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */
   MQHBAG adminBag = MQHB_UNUSABLE_HBAG;   /* admin bag for mqExecute         */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG cAttrsBag;                       /* bag containing chl attributes   */
   MQHBAG errorBag;                        /* bag containing cmd server error */
   MQLONG mqExecuteCC;                     /* mqExecute completion code       */
   MQLONG mqExecuteRC;                     /* mqExecute reason code           */
   MQLONG chlNameLength;                   /* Actual length of chl name       */
   MQLONG chlType;                         /* Channel type                    */
   MQLONG i;                               /* loop counter                    */
   MQLONG numberOfBags;                    /* number of bags in response bag  */
   MQCHAR chlName[MQ_OBJECT_NAME_LENGTH+1];/* name of chl extracted from bag  */
   MQCHAR OutputBuffer[100];               /* output data buffer              */
   OUTFILEHDL *outfp = NULL;               /* output file handle              */

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc &gt; 1)
      strncpy(qmName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(qmName, &hConn;, &compCode;, &connReason;);

   /***************************************************************************/
   /* Report the reason and stop if the connection failed.                    */
   /***************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("Queue Manager connection", compCode, connReason);
      exit( (int)connReason);
   }

   /***************************************************************************/
   /* Open the output file                                                    */
   /***************************************************************************/
   if (argc &gt; 2)
   {
     OPENOUTFILE(outfp, argv[2]);
   }
   else
   {
     OPENOUTFILE(outfp, OUTFILE);
   }

   if(outfp == NULL)
   {
     printf("Could not open output file.\n");
     goto MOD_EXIT;
   }
   /***************************************************************************/
   /* Create an admin bag for the mqExecute call                              */
   /***************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &adminBag;, &compCode;, &reason;);
   CheckCallResult("Create admin bag", compCode, reason);

   /***************************************************************************/
   /* Create a response bag for the mqExecute call                            */
   /***************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &responseBag;, &compCode;, &reason;);
   CheckCallResult("Create response bag", compCode, reason);

   /***************************************************************************/
   /* Put the generic channel name into the admin bag                         */
   /***************************************************************************/
   mqAddString(adminBag, MQCACH_CHANNEL_NAME, MQBL_NULL_TERMINATED, "*",
               &compCode;, &reason;);
   CheckCallResult("Add channel name", compCode, reason);

   /***************************************************************************/
   /* Put the channel type into the admin bag                                 */
```

```
/**************************************************************************/
mqAddInteger(adminBag, MQIACH_CHANNEL_TYPE, MQCHT_ALL, &compCode;, &reason;);
CheckCallResult("Add channel type", compCode, reason);

/**************************************************************************/
/* Add an inquiry for various attributes                                  */
/**************************************************************************/
mqAddInquiry(adminBag, MQIACH_CHANNEL_TYPE, &compCode;, &reason;);
CheckCallResult("Add inquiry", compCode, reason);

/**************************************************************************/
/* Send the command to find all the channel names and channel types.      */
/* The mqExecute call creates the PCF structure required, sends it to     */
/* the command server, and receives the reply from the command server into */
/* the response bag. The attributes are contained in system bags that are  */
/* embedded in the response bag, one set of attributes per bag.           */
/**************************************************************************/
mqExecute(hConn,                   /* MQ connection handle                */
          MQCMD_INQUIRE_CHANNEL,   /* Command to be executed              */
          MQHB_NONE,               /* No options bag                      */
          adminBag,                /* Handle to bag containing commands   */
          responseBag,             /* Handle to bag to receive the response*/
          MQHO_NONE,               /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,               /* Create a dynamic q for the response  */
          &compCode;,               /* Completion code from the mqexecute   */
          &reason;);                /* Reason code from mqexecute call      */

/**************************************************************************/
/* Check the command server is started. If not exit.                      */
/**************************************************************************/
if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName="">\n");
   goto MOD_EXIT;
}

/**************************************************************************/
/* Check the result from mqExecute call. If successful find the channel    */
/* types for all the channels. If failed find the error.                  */
/**************************************************************************/
if ( compCode == MQCC_OK )                      /* Successful mqExecute    */
{
  /************************************************************************/
  /* Count the number of system bags embedded in the response bag from the */
  /* mqExecute call. The attributes for each channel are in separate bags. */
  /************************************************************************/
  mqCountItems(responseBag, MQHA_BAG_HANDLE, &numberOfBags;,
             &compCode;, &reason;);
  CheckCallResult("Count number of bag handles", compCode, reason);

  for ( i=0; i<numberOfbags; i++)
  {
    /**********************************************************************/
    /* Get the next system bag handle out of the mqExecute response bag.   */
    /* This bag contains the channel attributes                           */
    /**********************************************************************/
    mqInquireBag(responseBag, MQHA_BAG_HANDLE, i, &cAttrsbag,
             &compCode, &reason);
    CheckCallResult("Get the result bag handle", compCode, reason);

    /**********************************************************************/
    /* Get the channel name out of the channel attributes bag             */
    /**********************************************************************/
    mqInquireString(cAttrsBag, MQCACH_CHANNEL_NAME, 0, MQ_OBJECT_NAME_LENGTH,
                chlName, &chlNameLength, NULL, &compCode, &reason);
    CheckCallResult("Get channel name", compCode, reason);
```

```
        /*******************************************************************/
        /* Get the channel type out of the channel attributes bag          */
        /*******************************************************************/

     mqInquireInteger(cAttrsBag, MQIACH_CHANNEL_TYPE, MQIND_NONE, &chlType,
                      &compCode, &reason);
        CheckCallResult("Get type", compCode, reason);

        /*******************************************************************/
        /* Use mqTrim to prepare the channel name for printing.            */
        /* Print the result.                                               */
        /*******************************************************************/
        mqTrim(MQ_CHANNEL_NAME_LENGTH, chlName, chlName, &compCode, &reason);
        sprintf(OutputBuffer, "%-20s%-9s", chlName, ChlType2String(chlType));
        WRITEOUTFILE(outfp,OutputBuffer,29)
      }
    }

    else                                               /* Failed mqExecute     */
    {
      printf("Call to get channel attributes failed: Cc = %ld : Rc = %ld\n",
                   compCode, reason);
      /*********************************************************************/
      /* If the command fails get the system bag handle out of the mqexecute   */
      /* response bag.This bag contains the reason from the command server     */
      /* why the command failed.                                           */
      /*********************************************************************/
      if (reason == MQRCCF_COMMAND_FAILED)
      {
        mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &errorBag,
                     &compCode, &reason);
        CheckCallResult("Get the result bag handle", compCode, reason);

        /*******************************************************************/
        /* Get the completion code and reason code, returned by the command    */
        /* server, from the embedded error bag.                            */
        /*******************************************************************/
        mqInquireInteger(errorBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                         &compCode, &reason );
        CheckCallResult("Get the completion code from the result bag",
                        compCode, reason);
        mqInquireInteger(errorBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                          &compCode, &reason);
        CheckCallResult("Get the reason code from the result bag",
                        compCode, reason);
        printf("Error returned by the command server: Cc = %ld : Rc = %ld\n",
                 mqExecuteCC, mqExecuteRC);
      }
    }

 MOD_EXIT:
    /***********************************************************************/
    /* Delete the admin bag if successfully created.                       */
    /***********************************************************************/
    if (adminBag != MQHB_UNUSABLE_HBAG)
    {
       mqDeleteBag(&adminBag, &compCode, &reason);
       CheckCallResult("Delete the admin bag", compCode, reason);
    }

    /***********************************************************************/
    /* Delete the response bag if successfully created.                    */
    /***********************************************************************/
    if (responseBag != MQHB_UNUSABLE_HBAG)
    {
       mqDeleteBag(&responseBag, &compCode, &reason);
       CheckCallResult("Delete the response bag", compCode, reason);
```

```
   }

   /**************************************************************************/
   /* Disconnect from the queue manager if not already connected            */
   /**************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("Disconnect from Queue Manager", compCode, reason);
   }

   /**************************************************************************/
   /* Close the output file if open                                          */
   /**************************************************************************/
   if(outfp != NULL)
      CLOSEOUTFILE(outfp);

   return 0;
}


/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the    */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %ld : Reason = %ld\n", callText,
                cc, rc);
}
```

Inquiring about queues and printing information (amqsailq.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAILQ.C                                                   */
/*                                                                            */
/* Description:  Sample C program to inquire the current depth of the local   */
/*               queues using the IBM MQ Administration Interface (MQAI)      */
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                         */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999, 2005                           */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAILQ is a sample C program that demonstrates how to inquire         */
/*    attributes of the local queue manager using the MQAI interface. In      */
/*    particular, it inquires the current depths of all the local queues.     */
/*                                                                            */
/*     - A PCF command is built by placing items into an MQAI administration  */
/*       bag.                                                                 */
/*       These are:-                                                          */
/*            - The generic queue name "*"                                    */
/*            - The type of queue required. In this sample we want to         */
/*              inquire local queues.                                         */
/*            - The attribute to be inquired. In this sample we want the      */
/*              current depths.                                               */
/*                                                                            */
/*     - The mqExecute call is executed with the command MQCMD_INQUIRE_Q.     */
/*       The call generates the correct PCF structure.                        */
/*       The default options to the call are used so that the command is sent */
/*       to the SYSTEM.ADMIN.COMMAND.QUEUE.                                   */
/*       The reply from the command server is placed on a temporary dynamic   */
/*       queue.                                                               */
/*       The reply from the MQCMD_INQUIRE_Q command is read from the          */
/*       temporary queue and formatted into the response bag.                 */
/*                                                                            */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server, then the code returned by      */
/*       command server is retrieved from the system bag that has been        */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                            */
/*     - If the call is successful, the depth of each local queue is placed   */
/*       in system bags embedded in the response bag of the mqExecute call.   */
/*       The name and depth of each queue is obtained from each of the bags   */
/*       and the result displayed on the screen.                              */
/*                                                                            */
/* Note: The command server must be running.                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAILQ has 1 parameter - the queue manager name (optional)               */
/*                                                                            */
/******************************************************************************/

/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);

/******************************************************************************/
/* Function: main                                                             */
/******************************************************************************/
int main(int argc, char *argv[])
{
   /***************************************************************************/
   /* MQAI variables                                                          */
   /***************************************************************************/
   MQHCONN hConn;                          /* handle to IBM MQ connection     */
   MQCHAR qmName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name             */
   MQLONG reason;                          /* reason code                     */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                          /* completion code                */
   MQHBAG adminBag = MQHB_UNUSABLE_HBAG;     /* admin bag for mqExecute        */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG qAttrsBag;                         /* bag containing q attributes     */
   MQHBAG errorBag;                          /* bag containing cmd server error */
   MQLONG mqExecuteCC;                       /* mqExecute completion code       */
   MQLONG mqExecuteRC;                       /* mqExecute reason code           */
   MQLONG qNameLength;                       /* Actual length of q name         */
   MQLONG qDepth;                            /* depth of queue                  */
   MQLONG i;                                 /* loop counter                    */
   MQLONG numberOfBags;                      /* number of bags in response bag  */
   MQCHAR qName[MQ_Q_NAME_LENGTH+1];         /* name of queue extracted from bag*/


   printf("Display current depths of local queues\n\n");

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 1)
      strncpy(qmName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(qmName, &hConn, &compCode, &connReason);

   /***************************************************************************/
   /* Report the reason and stop if the connection failed.                    */
   /***************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("Queue Manager connection", compCode, connReason);
      exit( (int)connReason);
   }

   /***************************************************************************/
   /* Create an admin bag for the mqExecute call                              */
   /***************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &adminBag, &compCode, &reason);
   CheckCallResult("Create admin bag", compCode, reason);
   /***************************************************************************/
   /* Create a response bag for the mqExecute call                            */
   /***************************************************************************/
   mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
   CheckCallResult("Create response bag", compCode, reason);

   /***************************************************************************/
   /* Put the generic queue name into the admin bag                           */
   /***************************************************************************/
   mqAddString(adminBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, "*",
               &compCode, &reason);
   CheckCallResult("Add q name", compCode, reason);

   /***************************************************************************/
   /* Put the local queue type into the admin bag                             */
   /***************************************************************************/
   mqAddInteger(adminBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
   CheckCallResult("Add q type", compCode, reason);

   /***************************************************************************/
   /* Add an inquiry for current queue depths                                 */
   /***************************************************************************/
   mqAddInquiry(adminBag, MQIA_CURRENT_Q_DEPTH, &compCode, &reason);
   CheckCallResult("Add inquiry", compCode, reason);

   /***************************************************************************/
   /* Send the command to find all the local queue names and queue depths.    */
   /* The mqExecute call creates the PCF structure required, sends it to      */
   /* the command server, and receives the reply from the command server into */
   /* the response bag. The attributes are contained in system bags that are  */
   /* embedded in the response bag, one set of attributes per bag.           */
   /**************************************************************************/
   mqExecute(hConn,                  /* IBM MQ connection handle            */
             MQCMD_INQUIRE_Q,        /* Command to be executed              */
             MQHB_NONE,              /* No options bag                      */
             adminBag,               /* Handle to bag containing commands   */
             responseBag,            /* Handle to bag to receive the response*/
             MQHO_NONE,              /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
             MQHO_NONE,              /* Create a dynamic q for the response */
             &compCode,              /* Completion code from the mqExecute  */
             &reason);               /* Reason code from mqExecute call     */


   /**************************************************************************/
   /* Check the command server is started. If not exit.                      */
   /**************************************************************************/
   if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
   {
      printf("Please start the command server: <strmqcsv QMgrName>\n");
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("Disconnect from Queue Manager", compCode, reason);
      exit(98);
   }

   /**************************************************************************/
   /* Check the result from mqExecute call. If successful find the current   */
   /* depths of all the local queues. If failed find the error.              */
   /**************************************************************************/
   if ( compCode == MQCC_OK )                      /* Successful mqExecute    */
   {
     /************************************************************************/
     /* Count the number of system bags embedded in the response bag from the */
     /* mqExecute call. The attributes for each queue are in a separate bag.  */
     /************************************************************************/
     mqCountItems(responseBag, MQHA_BAG_HANDLE, &numberOfBags, &compCode,
                  &reason);
     CheckCallResult("Count number of bag handles", compCode, reason);

     for ( i=0; i<numberOfBags; i++)
     {
       /**********************************************************************/
       /* Get the next system bag handle out of the mqExecute response bag.   */
       /* This bag contains the queue attributes                              */
       /**********************************************************************/
       mqInquireBag(responseBag, MQHA_BAG_HANDLE, i, &qAttrsBag, &compCode,
                    &reason);
       CheckCallResult("Get the result bag handle", compCode, reason);

       /**********************************************************************/
       /* Get the queue name out of the queue attributes bag                  */
       /**********************************************************************/
       mqInquireString(qAttrsBag, MQCA_Q_NAME, 0, MQ_Q_NAME_LENGTH, qName,
                       &qNameLength, NULL, &compCode, &reason);
       CheckCallResult("Get queue name", compCode, reason);

       /**********************************************************************/
       /* Get the depth out of the queue attributes bag                       */
       /**********************************************************************/
       mqInquireInteger(qAttrsBag, MQIA_CURRENT_Q_DEPTH, MQIND_NONE, &qDepth,
                        &compCode, &reason);
       CheckCallResult("Get depth", compCode, reason);

       /**********************************************************************/
       /* Use mqTrim to prepare the queue name for printing.                  */
       /* Print the result.                                                   */
       /**********************************************************************/
       mqTrim(MQ_Q_NAME_LENGTH, qName, qName, &compCode, &reason)
```

```
        printf("%4d  %-48s\n", qDepth, qName);
      }
    }

    else                                               /* Failed mqExecute     */
    {
      printf("Call to get queue attributes failed: Completion Code = %d :
              Reason = %d\n", compCode, reason);

      /*************************************************************************/
      /* If the command fails get the system bag handle out of the mqExecute   */
      /* response bag. This bag contains the reason from the command server    */
      /* why the command failed.                                               */
      /*************************************************************************/
      if (reason == MQRCCF_COMMAND_FAILED)
      {
        mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &errorBag, &compCode,
                     &reason);
        CheckCallResult("Get the result bag handle", compCode, reason);

       /************************************************************************/
       /* Get the completion code and reason code, returned by the command     */
       /* server, from the embedded error bag.                                 */
       /************************************************************************/
       mqInquireInteger(errorBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                        &compCode, &reason );
       CheckCallResult("Get the completion code from the result bag",
                        compCode, reason);
       mqInquireInteger(errorBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                         &compCode, &reason);
       CheckCallResult("Get the reason code from the result bag",
                        compCode, reason);
       printf("Error returned by the command server: Completion Code = %d :
               Reason = %d\n", mqExecuteCC, mqExecuteRC);
     }
   }

   /***************************************************************************/
   /* Delete the admin bag if successfully created.                           */
   /***************************************************************************/
   if (adminBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&adminBag, &compCode, &reason);
      CheckCallResult("Delete the admin bag", compCode, reason);
   }

   /***************************************************************************/
   /* Delete the response bag if successfully created.                        */
   /***************************************************************************/
   if (responseBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&responseBag, &compCode, &reason);
      CheckCallResult("Delete the response bag", compCode, reason);
   }

   /***************************************************************************/
   /* Disconnect from the queue manager if not already connected              */
   /***************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
       CheckCallResult("Disconnect from queue manager", compCode, reason);
   }
   return 0;
}

******************************************************************************/
```

```
*                                                                             */
* Function: CheckCallResult                                                   */
*                                                                             */
******************************************************************************/
*                                                                             */
* Input Parameters:  Description of call                                      */
*                    Completion code                                          */
*                    Reason code                                              */
*                                                                             */
* Output Parameters: None                                                     */
*                                                                             */
* Logic: Display the description of the call, the completion code and the     */
*        reason code if the completion code is not successful                 */
*                                                                             */
******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
  if (cc != MQCC_OK)
        printf("%s failed: Completion Code = %d : Reason = %d\n",
                callText, cc, rc);
}
```

Hints and tips for configuring IBM MQ

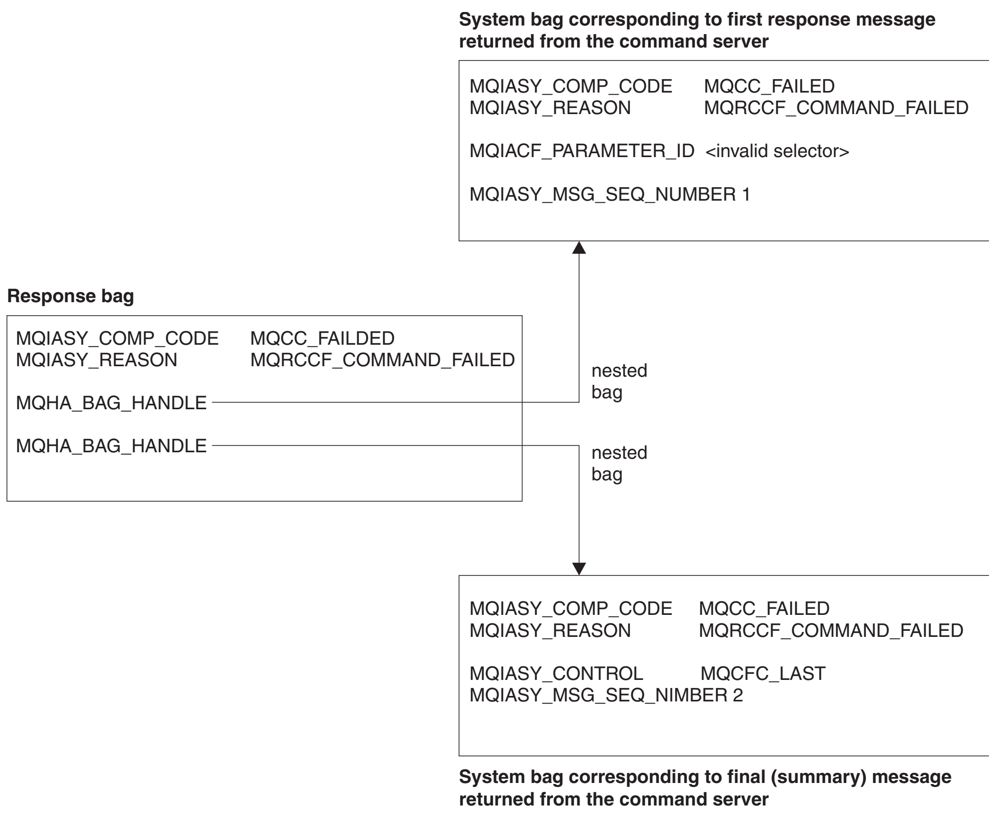
Programming hints and tips when using MQAI.

The MQAI uses PCF messages to send administration commands to the command server rather than dealing directly with the command server itself. Here are some tips for configuring IBM MQ using the MQAI:

- Character strings in IBM MQ are blank padded to a fixed length. Using C, null-terminated strings can normally be supplied as input parameters to IBM MQ programming interfaces.
- To clear the value of a string attribute, set it to a single blank rather than an empty string.
- Consider in advance the attributes that you want to change and inquire on just those attributes.
- Certain attributes cannot be changed, for example a queue name or a channel type. Ensure that you attempt to change only those attributes that can be modified. Refer to the list of required and optional parameters for the specific PCF change object. See Definitions of the Programmable Command Formats.
- If an MQAI call fails, some detail of the failure is returned to the response bag. Further detail can then be found in a nested bag that can be accessed by the selector MQHA_BAG_HANDLE. For example, if an mqExecute call fails with a reason code of MQRCCF_COMMAND_FAILED, this information is returned in the response bag. A possible reason for this reason code is that a selector specified was not valid for the type of command message and this detail of information is found in a nested bag that can be accessed by a bag handle.

  For more information on MQExecute, see "Sending administration commands to the command server using the mqExecute call" on page 54

  The following diagram shows this scenario:


System bag corresponding to first response message returned from the command server
MQIASY_COMP_CODE MQCC_FAILED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQIACF_PARAMETER_ID <invalid selector>
MQIASY_MSG_SEQ_NUMBER 1
Response bag
MQIASY_COMP_CODE MQCC_FAILDED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQHA_BAG_HANDLE
MQHA_BAG_HANDLE
nested bag
nested bag
MQIASY_COMP_CODE MQCC_FAILED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQIASY_CONTROL MQCFC_LAST
MQIASY_MSG_SEQ_NIMBER 2
System bag corresponding to final (summary) message returned from the command server

Advanced topics

Information on indexing, data conversion and use of message descriptor

- Indexing

  Indexes are used when replacing or removing existing data items from a bag to preserve insertion order. Full details on indexing can be found in Indexing.
- Data conversion

  The strings contained in an MQAI data bag can be in a variety of coded character sets and these can be converted using the mqSetInteger call. Full details on data conversion can be found in Data conversion.
- Use of the message descriptor

  MQAI generates a message descriptor which is set to an initial value when the data bag is created. Full details of the use of the message descriptor can be found in Use of the message descriptor.

Data bags

A data bag is a means of handling properties or parameters of objects using the MQAI.

Data Bags

- The data bag contains zero or more *data items*. These data items are ordered within the bag as they are placed into the bag. This is called the *insertion order*. Each data item contains a *selector* that identifies the data item and a *value* of that data item that can be either an integer, a 64-bit integer, an integer filter, a string, a string filter, a byte string, a byte string filter, or a handle of another bag. Data items are described in details in "Data item" on page 48

  There are two types of selector; *user selectors* and *system selectors*. These are described in MQAI Selectors. The selectors are usually unique, but it is possible to have multiple values for the same selector. In this case, an *index* identifies the particular occurrence of selector that is required. Indexes are described in Indexing.

  A hierarchy of these concepts is shown in Figure 1.

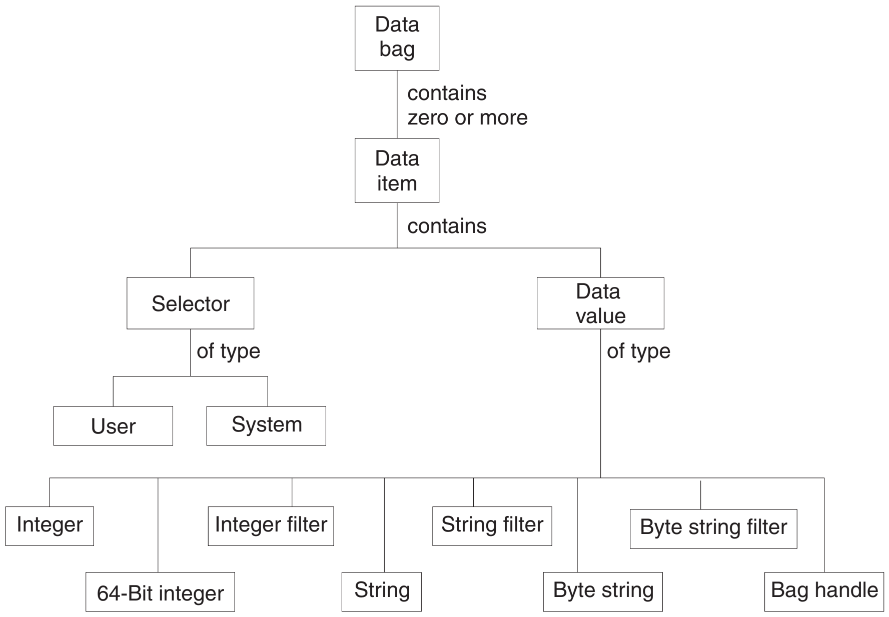


Figure 1. Hierarchy of MQAI concepts

The hierarchy has been explained in a previous paragraph.

Types of data bag

You can choose the type of data bag that you want to create depending on the task that you wish to perform:

user bag
A simple bag used for user data.

administration bag
A bag created for data used to administer IBM MQ objects by sending administration messages to a command server. The administration bag automatically implies certain options as described in "Creating and deleting data bags" on page 47.

command bag
A bag also created for commands for administering IBM MQ objects. However, unlike the

administration bag, the command bag does not automatically imply certain options although these options are available. For more information about options, see "Creating and deleting data bags."

group bag
A bag used to hold a set of grouped data items. Group bags cannot be used for administering IBM MQ objects.

In addition, the **system bag** is created by the MQAI when a reply message is returned from the command server and placed into a user's output bag. A system bag cannot be modified by the user.

Using Data Bags The different ways of using data bags are listed in this topic:

Using Data Bags

The different ways of using data bags are shown in the following list:
- You can create and delete data bags "Creating and deleting data bags."
- You can send data between applications using data bags "Putting and receiving data bags" on page 48.
- You can add data items to data bags "Adding data items to bags" on page 49.
- You can add an inquiry command within a data bag "Adding an inquiry command to a bag" on page 50.
- You can inquire within data bags "Inquiring within data bags" on page 51.
- You can count data items within a data bag "Counting data items" on page 53.
- You can change information within a data bag "Changing information within a bag" on page 51.
- You can clear a data bag "Clearing a bag using the mqClearBag call" on page 52.
- You can truncate a data bag "Truncating a bag using the mqTruncateBag call" on page 52.
- You can convert bags and buffers "Converting bags and buffers" on page 53.

Creating and deleting data bags:

Creating data bags

To use the MQAI, you first create a data bag using the mqCreateBag call. As input to this call, you supply one or more options to control the creation of the bag.

The `Options` parameter of the MQCreateBag call lets you choose whether to create a user bag, a command bag, a group bag, or an administration bag.

To create a user bag, a command bag, or a group bag, you can choose one or more further options to:
- Use the list form when there are two or more adjacent occurrences of the same selector in a bag.
- Reorder the data items as they are added to a PCF message to ensure that the parameters are in their correct order. For more information on data items, see "Data item" on page 48.
- Check the values of user selectors for items that you add to the bag.

Administration bags automatically imply these options.

A data bag is identified by its handle. The bag handle is returned from mqCreateBag and must be supplied on all other calls that use the data bag.

For a full description of the mqCreateBag call, see mqCreateBag.

Deleting data bags

Any data bag that is created by the user must also be deleted using the mqDeleteBag call. For example, if a bag is created in the user code, it must also be deleted in the user code.

System bags are created and deleted automatically by the MQAI. For more information about this, see "Sending administration commands to the command server using the mqExecute call" on page 54. User code cannot delete a system bag.

For a full description of the mqDeleteBag call, see mqDeleteBag.

Putting and receiving data bags:

Data can also be sent between applications by putting and getting data bags using the mqPutBag and mqGetBag calls. This lets the MQAI handle the buffer rather than the application. The mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue and the mqGetBag call removes the message from the specified queue and converts it back into a data bag. Therefore, the mqPutBag call is the equivalent of the mqBagToBuffer call followed by MQPUT, and the mqGetBag is the equivalent of the MQGET call followed by mqBufferToBag.

For more information on sending and receiving PCF messages in a specific queue, see "Sending and receiving PCF messages in a specified queue" on page 9

Note: If you choose to use the mqGetBag call, the PCF details within the message must be correct; if they are not, an appropriate error results and the PCF message is not returned.

Data item:

Data items are used to populate data bags when they are created. These data items can be user or system items.

These user items contain user data such as attributes of objects that are being administered. System items should be used for more control over the messages generated: for example, the generation of message headers. For more information about system items, see "System items" on page 49.

Types of Data Items

When you have created a data bag, you can populate it with integer or character-string items. You can inquire about all three types of item.

The data item can either be integer or character-string items. Here are the types of data item available within the MQAI:
- Integer
- 64-bit integer
- Integer filter
- Character-string
- String filter
- Byte string
- Byte string filter
- Bag handle

Using Data Items

These are the following ways of using data items:
- "Counting data items" on page 53.
- "Deleting data items" on page 53.
- "Adding data items to bags."
- "Filtering and querying data items" on page 50.

System items:

System items can be used for:
- The generation of PCF headers. System items can control the PCF command identifier, control options, message sequence number, and command type.
- Data conversion. System items handle the character-set identifier for the character-string items in the bag.

Like all data items, system items consist of a selector and a value. For information about these selectors and what they are for, see MQAI Selectors.

System items are unique. One or more system items can be identified by a system selector. There is only one occurrence of each system selector.

Most system items can be modified (see "Changing information within a bag" on page 51), but the bag-creation options cannot be changed by the user. You cannot delete system items. (See "Deleting data items" on page 53.)

Adding data items to bags:

When a data bag is created, you can populate it with data items. These data items can be user or system items. For more information about data items, see "Data item" on page 48.

The MQAI lets you add integer items, 64-bit integer items, integer filter items, character-string items, string filter, byte string items, and byte string filter items to bags and this is shown in Figure 2. The items are identified by a selector. Usually one selector identifies one item only, but this is not always the case. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag.

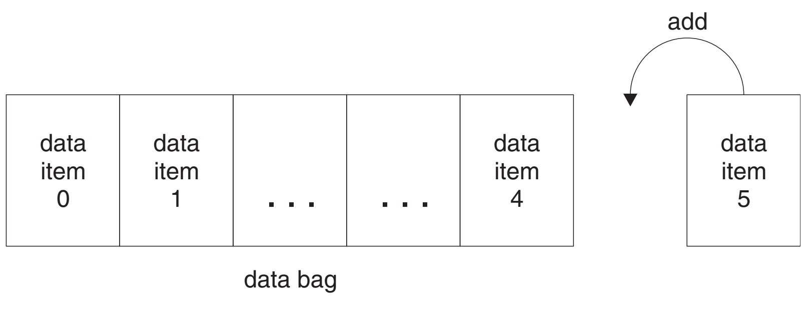


Figure 2. Adding data items

Add data items to a bag using the mqAdd* calls:
- To add integer items, use the mqAddInteger call as described in mqAddInteger
- To add 64-bit integer items, use the mqAddInteger64 call as described in mqAddInteger64
- To add integer filter items, use the mqAddIntegerFilter call as described in mqAddIntegerFilter
- To add character-string items, use the mqAddString call as described in mqAddString

- To add string filter items, use the mqAddStringFilter call as described in mqAddStringFilter
- To add byte string items, use the mqAddByteString call as described in mqAddByteString
- To add byte string filter items, use the mqAddByteStringFilter call as described in mqAddByteStringFilter

For more information on adding data items to a bag, see "System items" on page 49.

Adding an inquiry command to a bag:

The mqAddInquiry call is used to add an inquiry command to a bag. The call is specifically for administration purposes, so it can be used with administration bags only. It lets you specify the selectors of attributes on which you want to inquire from IBM MQ.

For a full description of the mqAddInquiry call, see mqAddInquiry.

Filtering and querying data items:

When using the MQAI to inquire about the attributes of IBM MQ objects, you can control the data that is returned to your program in two ways.

- You can ***filter*** the data that is returned using the mqAddInteger and mqAddString calls. This approach lets you specify a *Selector* and *ItemValue* pair, for example:
  ```
  mqAddInteger(inputbag, MQIA_Q_TYPE, MQQT_LOCAL)
  ```
  This example specifies that the queue type (*Selector*) must be local (*ItemValue*) and this specification must match the attributes of the object (in this case, a queue) about which you are inquiring.

  Other attributes that can be filtered correspond to the PCF Inquire* commands that can be found in "Introduction to Programmable Command Formats" on page 6. For example, to inquire about the attributes of a channel, see the Inquire Channel command in this product documentation. The "Required parameters" and "Optional parameters" of the Inquire Channel command identify the selectors that you can use for filtering.
- You can ***query*** particular attributes of an object using the mqAddInquiry call. This specifies the selector in which you are interested. If you do not specify the selector, all attributes of the object are returned.

Here is an example of filtering and querying the attributes of a queue:

```
/* Request information about all queues */
mqAddString(adminbag, MQCA_Q_NAME, "*")

/* Filter attributes so that local queues only are returned */
mqAddInteger(adminbag, MQIA_Q_TYPE, MQQT_LOCAL)

/* Query the names and current depths of the local queues */
mqAddInquiry(adminbag, MQCA_Q_NAME)
mqAddInquiry(adminbag, MQIA_CURRENT_Q_DEPTH)

/* Send inquiry to the command server and wait for reply */
mqExecute(MQCMD_INQUIRE_Q, ...)
```

For more examples of filtering and querying data items, see "Examples of using the MQAI" on page 19.

Inquiring within data bags:

You can inquire about:
- The value of an integer item using the mqInquireInteger call. See mqInquireInteger.
- The value of a 64-bit integer item using the mqInquireInteger64 call. See mqInquireInteger64.
- The value of an integer filter item using the mqInquireIntegerFilter call. See mqInquireIntegerFilter.
- The value of a character-string item using the mqInquireString call. See mqInquireString.
- The value of a string filter item using the mqInquireStringFilter call. See mqInquireStringFilter.
- The value of a byte string item using the mqInquireByteString call. See mqInquireByteString.
- The value of a byte string filter item using the mqInquireByteStringFilter call. See mqInquireByteStringFilter.
- The value of a bag handle using the mqInquireBag call. See mqInquireBag.

You can also inquire about the type (integer, 64-bit integer, integer filter, character string, string filter, byte string, byte string filter or bag handle) of a specific item using the mqInquireItemInfo call. See mqInquireItemInfo.

Changing information within a bag:

The MQAI lets you change information within a bag using the mqSet* calls. You can:
1. Modify data items within a bag. The index allows an individual instance of a parameter to be replaced by identifying the occurrence of the item to be modified (see Figure 3).

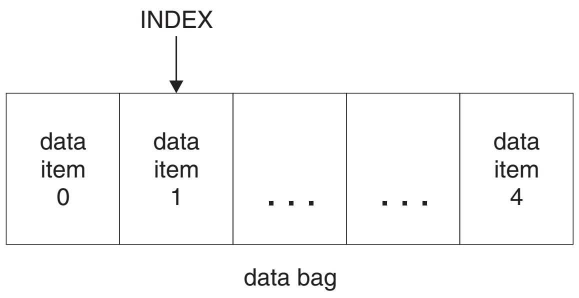


Figure 3. Modifying a single data item

2. Delete all existing occurrences of the specified selector and add a new occurrence to the end of the bag. (See Figure 4.) A special index value allows ***all*** instances of a parameter to be replaced.

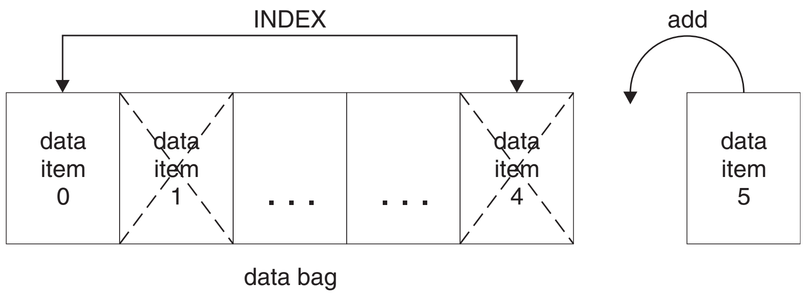


Figure 4. Modifying all data items

Note: The index preserves the insertion order within the bag but can affect the indices of other data items.

The mqSetInteger call lets you modify integer items within a bag. The mqSetInteger64 call lets you modify 64-bit integer items. The mqSetIntegerFilter call lets you modify integer filter items. The mqSetString call lets you modify character-string items. The mqSetStringFilter call lets you modify string filter items. The mqSetByteString call lets you modify byte string items. The mqSetByteStringFilter call lets you modify byte string filter items. Alternatively, you can use these calls to delete all existing occurrences of the specified selector and add a new occurrence at the end of the bag. The data item can be a user item or a system item.

For a full description of these calls, see:
- mqSetInteger
- mqSetInteger64
- mqSetIntegerFilter
- mqSetString
- mqSetStringFilter
- mqSetByteString
- mqSetByteStringFilter

Clearing a bag using the mqClearBag call:

The mqClearBag call removes all user items from a user bag and resets system items to their initial values. System bags contained within the bag are also deleted.

For a full description of the mqClearBag call, see mqClearBag.

Truncating a bag using the mqTruncateBag call:

The mqTruncateBag call reduces the number of user items in a user bag by deleting the items from the end of the bag, starting with the most recently added item. For example, it can be used when using the same header information to generate more than one message.

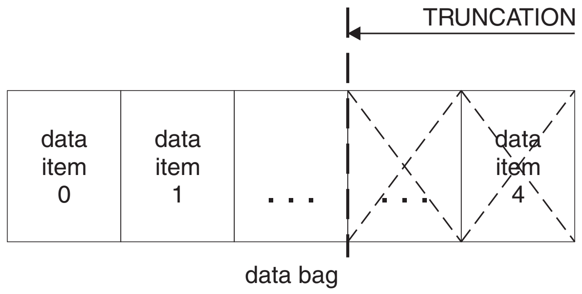


Figure 5. Truncating a bag

For a full description of the mqTruncateBag call, see mqTruncateBag.

Converting bags and buffers:

To send data between applications, firstly the message data is placed in a bag. Then, the data in the bag is converted into a PCF message using the mqBagToBuffer call. The PCF message is sent to the required queue using the MQPUT call. This is shown in Figure Figure 6. For a full description of the mqBagToBuffer call, see mqBagToBuffer.

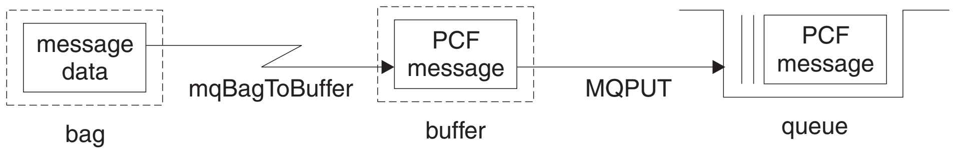


Figure 6. Converting bags to PCF messages

To receive data, the message is received into a buffer using the MQGET call. The data in the buffer is then converted into a bag using the mqBufferToBag call, providing the buffer contains a valid PCF message. This is shown in Figure Figure 7. For a full description of the mqBufferToBag call, see mqBufferToBag.

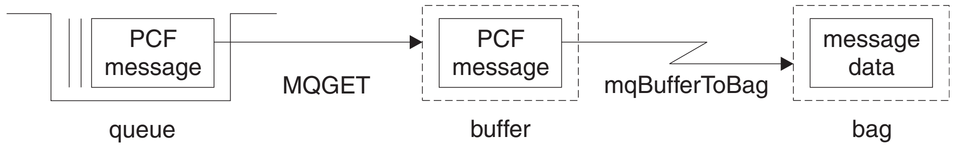


Figure 7. Converting PCF messages to bag form

Counting data items: The mqCountItems call counts the number of user items, system items, or both, that are stored in a data bag, and returns this number. For example, `mqCountItems( Bag, 7, ...)`, returns the number of items in the bag with a selector of 7. It can count items by individual selector, by user selectors, by system selectors, or by all selectors.

Note: This call counts the number of data items, not the number of unique selectors in the bag. A selector can occur multiple times, so there might be fewer unique selectors in the bag than data items.

For a full description of the mqCountItems call, see mqCountItems.

Deleting data items:

You can delete items from bags in a number of ways. You can:
- Remove one or more user items from a bag. For detailed information, see “Deleting data items from a bag using the mqDeleteItem call” on page 54.
- Delete ***all*** user items from a bag, that is, *clear* a bag. For detailed information see “Clearing a bag using the mqClearBag call” on page 52.
- Delete user items from the end of a bag, that is, *truncate* a bag. For detailed information, see “Truncating a bag using the mqTruncateBag call” on page 52.

Deleting data items from a bag using the mqDeleteItem call:

The mqDeleteItem call removes one or more user items from a bag. The index is used to delete either:

1. A single occurrence of the specified selector. (See Figure 8.)

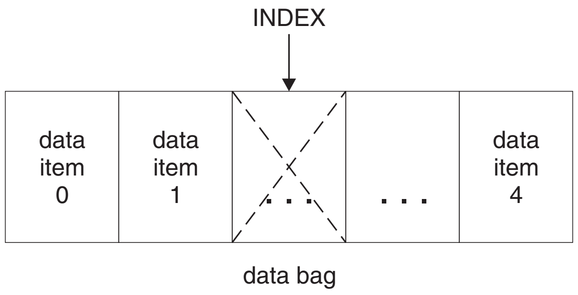


Figure 8. Deleting a single data item

or

2. All occurrences of the specified selector. (See Figure 9.)

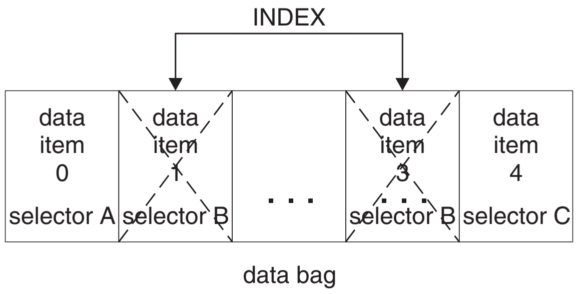


Figure 9. Deleting all data items

Note: The index preserves the insertion order within the bag but can affect the indices of other data items. For example, the mqDeleteItem call does not preserve the index values of the data items that follow the deleted item because the indices are reorganized to fill the gap that remains from the deleted item.

For a full description of the mqDeleteItem call, see mqDeleteItem.

Sending administration commands to the command server using the mqExecute call

When a data bag has been created and populated, an administrative command message can be sent to the command server of a queue manager using the mqExecute call. This handles the exchange with the command server and returns responses in a bag.

After you have created and populated your data bag, you can send an administration command message to the command server of a queue manager. The easiest way to do this is by using the mqExecute call. The mqExecute call sends an administration command message as a nonpersistent message and waits for any responses. Responses are returned in a response bag. These might contain information about attributes relating to several IBM MQ objects or a series of PCF error response messages, for example. Therefore, the response bag could contain a return code only or it could contain *nested bags*.

Response messages are placed into system bags that are created by the system. For example, for inquiries about the names of objects, a system bag is created to hold those object names and the bag is inserted into the user bag. Handles to these bags are then inserted into the response bag and the nested bag can be accessed by the selector MQHA_BAG_HANDLE. The system bag stays in storage, if it is not deleted, until the response bag is deleted.

The concept of *nesting* is shown in Figure 10.

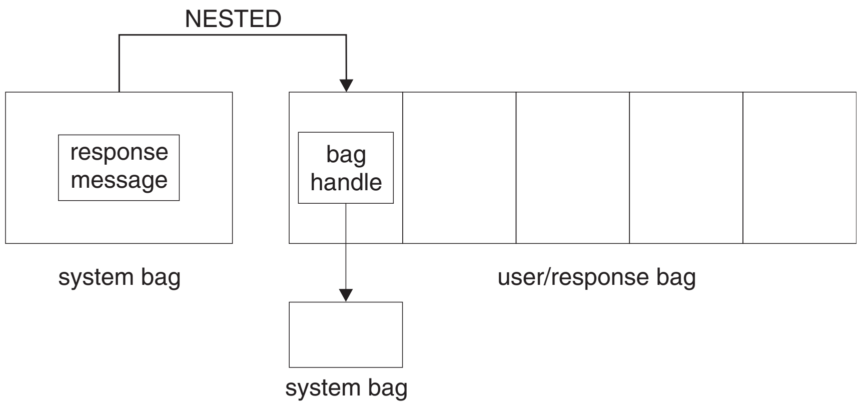


Figure 10. Nesting

As input to the mqExecute call, you must supply:
- An MQI connection handle.
- The command to be executed. This should be one of the MQCMD_* values.

  Note: If this value is not recognized by the MQAI, the value is still accepted. However, if the mqAddInquiry call was used to insert values into the bag, this parameter must be an INQUIRE command recognized by the MQAI. That is, the parameter should be of the form MQCMD_INQUIRE_*.
- Optionally, a handle of the bag containing options that control the processing of the call. This is also where you can specify the maximum time in milliseconds that the MQAI should wait for each reply message.
- A handle of the administration bag that contains details of the administration command to be issued.
- A handle of the response bag that receives the reply messages.

The following are optional:
- An object handle of the queue where the administration command is to be placed.

  If no object handle is specified, the administration command is placed on the SYSTEM.ADMIN.COMMAND.QUEUE belonging to the currently connected queue manager. This is the default.
- An object handle of the queue where reply messages are to be placed.

  You can choose to place the reply messages on a dynamic queue that is created automatically by the MQAI. The queue created exists for the duration of the call only, and is deleted by the MQAI on exit from the mqExecute call.

For examples uses of the mqExecute call, see Example code

Administration using the MQ Explorer

The MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows, or Linux x86-64 only.

IBM MQ for Windows and IBM MQ for Linux x86-64 provide an administration interface called the MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands. Comparing command sets shows you what you can do using the MQ Explorer.

The MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows, or Linux x86-64, by pointing the MQ Explorer at the queue managers and clusters you are interested in. The platforms and levels of IBM MQ that can be administered using the MQ Explorer are described in "Remote queue managers" on page 57.

To configure remote IBM MQ queue managers so that MQ Explorer can administer them, see "Prerequisite software and definitions" on page 58.

It allows you to perform tasks, typically associated with setting up and fine-tuning the working environment for IBM MQ, either locally or remotely within a Windows or Linux x86-64 system domain.

On Linux, the MQ Explorer might fail to start if you have more than one Eclipse installation. If this happens, start the MQ Explorer using a different user ID to the one you use for the other Eclipse installation.

On Linux, to start the MQ Explorer successfully, you must be able to write a file to your home directory, and the home directory must exist.

What you can do with the IBM MQ Explorer

This is a list of the tasks that you can perform using the IBM MQ Explorer.

With the IBM MQ Explorer, you can:

- Create and delete a queue manager (on your local machine only).
- Start and stop a queue manager (on your local machine only).
- Define, display, and alter the definitions of IBM MQ objects such as queues and channels.
- Browse the messages on a queue.
- Start and stop a channel.
- View status information about a channel, listener, queue, or service objects.
- View queue managers in a cluster.
- Check to see which applications, users, or channels have a particular queue open.
- Create a new queue manager cluster using the *Create New Cluster* wizard.
- Add a queue manager to a cluster using the *Add Queue Manager to Cluster* wizard.
- Manage the authentication information object, used with Secure Sockets Layer (SSL) channel security.
- Create and delete channel initiators, trigger monitors, and listeners.
- Start or stop the command servers, channel initiators, trigger monitors, and listeners.
- Set specific services to start automatically when a queue manager is started.
- Modify the properties of queue managers.
- Change the local default queue manager.
- Invoke the ikeyman GUI to manage secure sockets layer (SSL) certificates, associate certificates with queue managers, and configure and setup certificate stores (on your local machine only).
- Create JMS objects from IBM MQ objects, and IBM MQ objects from JMS objects.
- Create a JMS Connection Factory for any of the currently supported types.

- Modify the parameters for any service, such as the TCP port number for a listener, or a channel initiator queue name.
- Start or stop the service trace.

You perform administration tasks using a series of *Content Views* and *Property dialogs*.

Content View
A Content View is a panel that can display the following:
- Attributes, and administrative options relating to IBM MQ itself.
- Attributes, and administrative options relating to one or more related objects.
- Attributes, and administrative options for a cluster.

Property dialogs
A property dialog is a panel that displays attributes relating to an object in a series of fields, some of which you can edit.

You navigate through the IBM MQ Explorer using the *Navigator view*. The Navigator allows you to select the Content View you require.

Remote queue managers

There are two exceptions to the supported queue managers that you can connect to.

From a Windows or Linux (x86 and x86-64 platforms) system, the IBM MQ Explorer can connect to all supported queue managers with the following exceptions:
- IBM MQ for z/OS queue managers earlier than Version 6.0.
- Currently supported MQSeries® V2 queue managers.

The IBM MQ Explorer handles the differences in the capabilities between the different command levels and platforms. However, if it encounters an attribute that it does not recognize, the attribute will not be visible.

If you intend to remotely administer a V6.0 or later queue manager on Windows using the IBM MQ Explorer on an IBM MQ V5.3 computer, you must install Fix Pack 9 (CSD9) or later on your IBM MQ for Windows V5.3 computer.

If you intend to remotely administer a V5.3 queue manager on iSeries using the IBM MQ Explorer on an IBM MQ V6.0 or later computer, you must install Fix Pack 11 (CSD11) or later on your IBM MQ for iSeries V5.3 computer. This fix pack corrects connection problems between the IBM MQ Explorer and the iSeries queue manager.

Deciding whether to use the IBM MQ Explorer

When deciding whether to use the IBM MQ Explorer at your installation, consider the information listed in this topic.

You need to be aware of the following points:

Object names
: If you use lowercase names for queue managers and other objects with the IBM MQ Explorer, when you work with the objects using MQSC commands, you must enclose the object names in single quotation marks, or IBM MQ does not recognize them.

Large queue managers
: The IBM MQ Explorer works best with small queue managers. If you have a large number of objects on a single queue manager, you might experience delays while the IBM MQ Explorer extracts the required information to present in a view.

Clusters
: IBM MQ clusters can potentially contain hundreds or thousands of queue managers. The IBM MQ Explorer presents the queue managers in a cluster using a tree structure. The physical size of a cluster does not affect the speed of the IBM MQ Explorer dramatically because the IBM MQ Explorer does not connect to the queue managers in the cluster until you select them.

Setting up the IBM MQ Explorer

This section outlines the steps you need to take to set up the IBM MQ Explorer.

- "Prerequisite software and definitions"
- "Security" on page 59
- "Showing and hiding queue managers and clusters" on page 62
- "Cluster membership" on page 63
- "Data conversion" on page 64

Prerequisite software and definitions

Ensure that you satisfy the following requirements before trying to use the MQ Explorer.

The MQ Explorer can connect to remote queue managers using the TCP/IP communication protocol only.

Check that:

1. A command server is running on every remotely administered queue manager.
2. A suitable TCP/IP listener object must be running on every remote queue manager. This object can be the IBM MQ listener or, on UNIX and Linux systems, the inetd daemon.
3. A server-connection channel, by default named SYSTEM.ADMIN.SVRCONN, exists on all remote queue managers.

   You can create the channel using the following MQSC command:

   ```
   DEFINE CHANNEL(SYSTEM.ADMIN.SVRCONN) CHLTYPE(SVRCONN)
   ```

   This command creates a basic channel definition. If you want a more sophisticated definition (to set up security, for example), you need additional parameters. For more information, see DEFINE CHANNEL.
4. The system queue, SYSTEM.MQEXPLORER.REPLY.MODEL, must exist.

Security

If you are using IBM MQ in an environment where it is important for you to control user access to particular objects, you might need to consider the security aspects of using the IBM MQ Explorer.

Authorization to use the IBM MQ Explorer:

Any user can use the IBM MQ Explorer, but certain authorities are required to connect, access, and manage queue managers.

To perform local administrative tasks using the IBM MQ Explorer, a user is required to have the necessary authority to perform the administrative tasks. If the user is a member of the mqm group, the user has authority to perform all local administrative tasks.

To connect to a remote queue manager and perform remote administrative tasks using the IBM MQ Explorer, the user executing the IBM MQ Explorer is required to have the following authorities:
- CONNECT authority on the target queue manager object
- INQUIRE authority on the target queue manager object
- DISPLAY authority to the target queue manager object
- INQUIRE authority to the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- DISPLAY authority to the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- INPUT (get) authority to the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- OUTPUT (put) authority to the queue, SYSTEM.ADMIN.COMMAND.QUEUE
- INQUIRE authority on the queue, SYSTEM.ADMIN.COMMAND.QUEUE
- Authority to perform the action selected

Note: INPUT authority relates to input to the user from a queue (a get operation). OUTPUT authority relates to output from the user to a queue (a put operation).

To connect to a remote queue manager on IBM MQ for z/OS and perform remote administrative tasks using the IBM MQ Explorer, the following must be provided:
- A RACF® profile for the system queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- A RACF profile for the queues, AMQ.MQEXPLORER.*

In addition, the user executing the IBM MQ Explorer is required to have the following authorities:
- RACF UPDATE authority to the system queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- RACF UPDATE authority to the queues, AMQ.MQEXPLORER.*
- CONNECT authority on the target queue manager object
- Authority to perform the action selected
- READ authority to all the hlq.DISPLAY.object profiles in the MQCMDS class

For information about how to grant authority to IBM MQ objects, see Giving access to an IBM MQ object on UNIX or Linux systems and Windows .

If a user attempts to perform an operation that they are not authorized to perform, the target queue manager invokes authorization failure procedures and the operation fails.

The default filter in the IBM MQ Explorer is to display all IBM MQ objects. If there are any IBM MQ objects that a user does not have DISPLAY authority to, authorization failures are generated. If authority events are being recorded, restrict the range of objects that are displayed to those objects that the user has DISPLAY authority to.

Security for connecting to remote queue managers:

You must secure the channel between the IBM MQ Explorer and each remote queue manager.

The IBM MQ Explorer connects to remote queue managers as an MQI client application. This means that each remote queue manager must have a definition of a server-connection channel and a suitable TCP/IP listener. If you do not secure your server connection channel it is possible for a malicious application to connect to the same server connection channel and gain access to the queue manager objects with unlimited authority. In order to secure your server connection channel either specify a non-blank value for the MCAUSER attribute of the channel, use channel authentication records, or use a security exit.

The default value of the MCAUSER attribute is the local user ID. If you specify a non-blank user name as the MCAUSER attribute of the server connection channel, all programs connecting to the queue manager using this channel run with the identity of the named user and have the same level of authority. This does not happen if you use channel authentication records.

Using a security exit with the IBM MQ Explorer:

You can specify a default security exit and queue manager specific security exits using the IBM MQ Explorer.

You can define a default security exit, which can be used for all new client connections from the IBM MQ Explorer. This default exit can be overridden at the time a connection is made. You can also define a security exit for a single queue manager or a set of queue managers, which takes effect when a connection is made. You specify exits using the IBM MQ Explorer. For more information, see the IBM MQ Help Center.

Using the MQ Explorer to connect to a remote queue manager using SSL-enabled MQI channels:

The MQ Explorer connects to remote queue managers using an MQI channel. If you want to secure the MQI channel using SSL security, you must establish the channel using a client channel definition table.

For information how to establish an MQI channel using a client channel definition table, see Overview of IBM MQ MQI clients.

When you have established the channel using a client channel definition table, you can use the MQ Explorer to connect to a remote queue manager using SSL-enabled MQI channel, as described in "Tasks on the system that hosts the remote queue manager" and "Tasks on the system that hosts the MQ Explorer" on page 61.

Tasks on the system that hosts the remote queue manager

On the system hosting the remote queue manager, perform the following tasks:

1. Define a server connection and client connection pair of channels, and specify the appropriate value for the *SSLCIPH* variable on the server connection on both channels. For more information about the *SSLCIPH* variable, see Protecting channels with SSL
2. Send the channel definition table `AMQCLCHL.TAB`, which is found in the queue manager's `@ipcc` directory, to the system hosting the MQ Explorer.
3. Start a TCP/IP listener on a designated port.
4. Place both the CA and personal SSL certificates into the SSL directory of the queue manager:
   - `/var/mqm/qmgrs/+QMNAME+/SSL` for UNIX and Linux systems
   - `C:\Program Files\IBM\WebSphere MQ\qmgrs\+QMNAME+\SSL` for Windows systems

   Where `+QMNAME+` is a token representing the name of the queue manager.

5. Create a key database file of type CMS named `key.kdb`. Stash the password in a file either by checking the option in the iKeyman GUI, or by using the `-stash` option with the **`runmqckm`** commands.
6. Add the CA certificates to the key database created in the previous step.
7. Import the personal certificate for the queue manager into the key database.

For more detailed information about working with the Secure Sockets Layer on Windows systems, see Working with SSL or TLS on UNIX, Linux and Windows systems.

Tasks on the system that hosts the MQ Explorer

On the system hosting the MQ Explorer, perform the following tasks:

1. Create a key database file of type JKS named `key.jks`. Set a password for this key database file.

   The MQ Explorer uses Java keystore files (JKS) for SSL security, and so the keystore file being created for configuring SSL for the MQ Explorer must match this.
2. Add the CA certificates to the key database created in the previous step.
3. Import the personal certificate for the queue manager into the key database.
4. On Windows and Linux systems, start MQ Explorer by using the system menu, the `MQExplorer` executable file, or the **`strmqcfg`** command.
5. From the MQ Explorer toolbar, click **Window -> Preferences**, then expand **IBM MQ Explorer** and click **SSL Client Certificate Stores**. Enter the name of, and password for, the JKS file created in step 1 of "Tasks on the system that hosts the MQ Explorer," in both the Trusted Certificate Store and the Personal Certificate Store, then click **OK**.
6. Close the **Preferences** window, and right-click **Queue Managers**. Click **Show/Hide Queue Managers**, and then click **Add** on the **Show/Hide Queue Managers** screen.
7. Type the name of the queue manager, and select the **Connect directly** option. Click next.
8. Select **Use client channel definition table (CCDT)** and specify the location of the channel table file that you transferred from the remote queue manager in step 2 in "Tasks on the system that hosts the remote queue manager" on page 60 on the system hosting the remote queue manager.
9. Click **Finish**. You can now access the remote queue manager from the MQ Explorer.

Connecting through another queue manager:

The IBM MQ Explorer allows you to connect to a queue manager through an intermediate queue manager, to which the IBM MQ Explorer is already connected.

In this case, the IBM MQ Explorer puts PCF command messages to the intermediate queue manager, specifying the following:

- The *ObjectQMgrName* parameter in the object descriptor (MQOD) as the name of the target queue manager. For more information on queue name resolution, see the Name resolution.
- The *UserIdentifier* parameter in the message descriptor (MQMD) as the local userId.

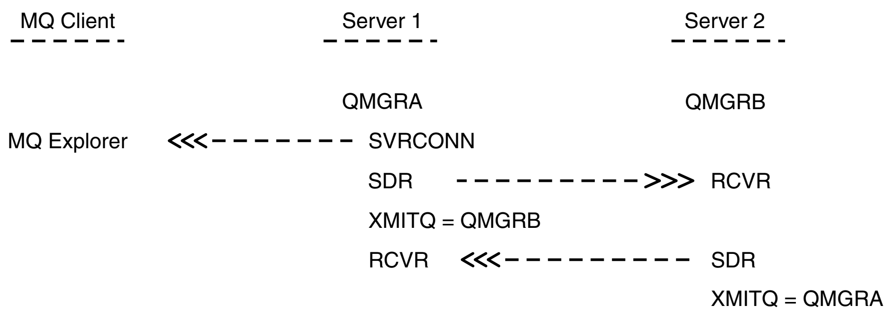
If the connection is then used to connect to the target queue manager via an intermediate queue manager, the userId is flowed in the *UserIdentifier* parameter of the message descriptor (MQMD) again. In order for the MCA listener on the target queue manager to accept this message, either the MCAUSER attribute must be set, or the userId must already exist with put authority.

The command server on the target queue manager puts messages to the transmission queue specifying the userId in the *UserIdentifier* parameter in the message descriptor (MQMD). For this put to succeed the userId must already exist on the target queue manager with put authority.

The following example shows you how to connect a queue manager, through an intermediate queue manager, to the IBM MQ Explorer.

Establish a remote administration connection to a queue manager. Verify that the:

- Queue manager on the server is active and has a server-connection channel (SVRCONN) defined.
- Listener is active.
- Command server is active.
- SYSTEM.MQ EXPLORER.REPLY.MODEL queue has been created and that you have sufficient authority.
- Queue manager listeners, command servers, and sender channels are started.



In this example:

- IBM MQ Explorer is connected to queue manager QMGRA (running on Server1) using a client connection.
- Queue manager QMGRB on Server2 can be now connected to IBM MQ Explorer through an intermediate queue manager (QMGRA)
- When connecting to QMGRB with IBM MQ Explorer, select QMGRA as the intermediate queue manager

In this situation, there is no direct connection to QMGRB from IBM MQ Explorer; the connection to QMGRB is through QMGRA.

Queue manager QMGRB on Server2 is connected to QMGRA on Server1 using sender-receiver channels. The channel between QMGRA and QMGRB must be set up in such a way that remote administration is possible; see "Preparing channels and transmission queues for remote administration" on page 133.

Showing and hiding queue managers and clusters

The IBM MQ Explorer can display more than one queue manager at a time. From the Show/Hide Queue Manager panel (selectable from the menu for the Queue Managers tree node), you can choose whether you display information about another (remote) machine. Local queue managers are detected automatically.

To show a remote queue manager:

1. Right-click the **Queue Managers** tree node, then select *Show/Hide Queue Managers...*.
2. Click **Add**. The Show/Hide Queue Managers panel is displayed.
3. Enter the name of the remote queue manager and the host name or IP address in the fields provided.

   The host name or IP address is used to establish a client connection to the remote queue manager using either its default server connection channel, SYSTEM.ADMIN.SVRCONN, or a user-defined server connection channel.
4. Click **Finish**.

The Show/Hide Queue Managers panel also displays a list of all visible queue managers. You can use this panel to hide queue managers from the navigation view.

If the IBM MQ Explorer displays a queue manager that is a member of a cluster, the cluster is detected, and displayed automatically.

To export the list of remote queue managers from this panel:
1. Close the Show/Hide Queue Managers panel.
2. Right-click the top **IBM MQ** tree node in the Navigation pane of the IBM MQ Explorer, then select **Export MQ Explorer Settings**
3. Click **MQ Explorer > MQ Explorer Settings**
4. Select **Connection Information > Remote queue managers**.
5. Select a file to store the exported settings in.
6. Finally, click **Finish** to export the remote queue manager connection information to the specified file.

To import a list of remote queue managers:
1. Right-click the top **IBM MQ** tree node in the Navigation pane of the IBM MQ Explorer, then select **Import MQ Explorer Settings**
2. Click **MQ Explorer > MQ Explorer Settings**
3. Click **Browse**, and navigate to the path of the file that contains the remote queue manager connection information.
4. Click **Open**. If the file contains a list of remote queue managers, the **Connection Information > Remote queue managers** box is selected.
5. Finally, click **Finish** to import the remote queue manager connection information into the IBM MQ Explorer.

Cluster membership

IBM MQ Explorer requires information about queue managers that are members of a cluster.

If a queue manager is a member of a cluster, then the cluster tree node will be populated automatically.

If queue managers become members of clusters while the IBM MQ Explorer is running, then you must maintain the IBM MQ Explorer with up-to-date administration data about clusters so that it can communicate effectively with them and display correct cluster information when requested. In order to do this, the IBM MQ Explorer needs the following information:
- The name of a repository queue manager
- The connection name of the repository queue manager if it is on a remote queue manager

With this information, the IBM MQ Explorer can:
- Use the repository queue manager to obtain a list of queue managers in the cluster.
- Administer the queue managers that are members of the cluster and are on supported platforms and command levels.

Administration is not possible if:
- The chosen repository becomes unavailable. The IBM MQ Explorer does not automatically switch to an alternative repository.
- The chosen repository cannot be contacted over TCP/IP.
- The chosen repository is running on a queue manager that is running on a platform and command level not supported by the IBM MQ Explorer.

The cluster members that can be administered can be local, or they can be remote if they can be contacted using TCP/IP. The IBM MQ Explorer connects to local queue managers that are members of a cluster directly, without using a client connection.

Data conversion

The IBM MQ Explorer works in CCSID 1208 (UTF-8). This enables the IBM MQ Explorer to display the data from remote queue managers correctly. Whether connecting to a queue manager directly, or by using an intermediate queue manager, the IBM MQ Explorer requires all incoming messages to be converted to CCSID 1208 (UTF-8).

An error message is issued if you try to establish a connection between the IBM MQ Explorer and a queue manager with a CCSID that the IBM MQ Explorer does not recognize.

Supported conversions are described in Code page conversion.

Security on Windows

The Prepare IBM MQ wizard creates a special user account so that the Windows service can be shared by processes that need to use it.

A Windows service is shared between client processes for an IBM MQ installation. One service is created for each installation. Each service is named `MQ_`*`InstallationName`*, and has a display name of `IBM MQ(`*`InstallationName`*`)`. Before Version 7.1, with only one installation on a server the single, Windows service was named MQSeriesServices with the display name `MQSeries`.

Because each service must be shared between non-interactive and interactive logon sessions, you must launch each under a special user account. You can use one special user account for all the services, or create different special user accounts. Each special user account must have the user right to "Logon as a service", for more information see "User rights required for an IBM MQ Windows Service" on page 65

When you install IBM MQ and run the Prepare IBM MQ wizard for the first time, it creates a local user account for the service called MUSR_MQADMIN with the required settings and permissions, including "Logon as a service".

For subsequent installations, the Prepare IBM MQ wizard creates a user account named MUSR_MQADMINx, where x is the next available number representing a user ID that does not exist. The password for MUSR_MQADMINx is randomly generated when the account is created, and used to configure the logon environment for the service. The generated password does not expire.

This IBM MQ account is not affected by any account policies that are set up on the system to require that account passwords are changed after a certain period.

The password is not known outside this one-time processing and is stored by the Windows operating system in a secure part of the registry.

Related information:

Windows: "Logon as a service" required

Using Active directory (Windows only)

In some network configurations, where user accounts are defined on domain controllers that are using Active Directory, the local user account IBM MQ is running under might not have the authority it requires to query the group membership of other domain user accounts. The Prepare IBM MQ Wizard identifies whether this is the case by carrying out tests and asking the user questions about the network configuration.

If the local user account IBM MQ is running under does not have the required authority, the Prepare IBM MQ Wizard prompts the user for the account details of a domain user account with particular user rights. For the user rights that the domain user account requires see "User rights required for an IBM MQ Windows Service." Once the user has entered valid account details for the domain user account into the Prepare IBM MQ Wizard, it configures an IBM MQ Windows service to run under the new account. The account details are held in the secure part of the Registry and cannot be read by users.

When the service is running, an IBM MQ Windows service is launched and remains running for as long as the service is running. An IBM MQ administrator who logs on to the server after the Windows service is launched can use the MQ Explorer to administer queue managers on the server. This connects the MQ Explorer to the existing Windows service process. These two actions need different levels of permission before they can work:

- The launch process requires a launch permission.
- The IBM MQ administrator requires Access permission.

User rights required for an IBM MQ Windows Service

The table in this topic lists the user rights required for the local and domain user account under which the Windows service for an IBM MQ installation runs.

Log on as batch job	Enables an IBM MQ Windows service to run under this user account.
Log on as service	Enables users to set the IBM MQ Windows service to log on using the configured account.
Shut down the system	Allows the IBM MQ Windows service to restart the server if configured to do so when recovery of a service fails.
Increase quotas	Required for operating system **CreateProcessAsUser** call.
Act as part of the operating system	Required for operating system **LogonUser** call.
Bypass traverse checking	Required for operating system **LogonUser** call.
Replace a process level token	Required for operating system **LogonUser** call.

Note: Debug programs rights might be needed in environments running ASP and IIS applications. Your domain user account must have these Windows user rights set as effective user rights as listed in the Local Security Policy application. If they are not, set them using either the Local Security Policy application locally on the server, or by using the Domain Security Application domain wide.

Changing the user name associated with the IBM MQ Service

You might need to change the user name associated with the IBM MQ Service from MUSR_MQADMIN to something else. (For example, you might need to do this if your queue manager is associated with DB2®, which does not accept user names of more than 8 characters.)

Procedure

1. Create a new user account (for example **NEW_NAME**)
2. Use the Prepare IBM MQ Wizard to enter the details of the new user account.

Changing the password of the IBM MQ Windows service user account

About this task

To change the password of the IBM MQ Windows service local user account, perform the following steps:

Procedure

1. Identify the user the service is running under.
2. Stop the IBM MQ service from the Computer Management panel.
3. Change the required password in the same way that you would change the password of an individual.
4. Go to the properties for the IBM MQ service from the Computer Management panel.
5. Select the **Log On** Page.
6. Confirm that the account name specified matches the user for which the password was modified.
7. Type the password into the **Password** and **Confirm password** fields and click **OK**.

IBM MQ Windows service for an installation running under a domain user account:

About this task

If the IBM MQ Windows service for an installation is running under a domain user account, you can also change the password for the account as follows:

Procedure

1. Change the password for the domain account on the domain controller. You might need to ask your domain administrator to do this for you.
2. Follow the steps to modify the **Log On** page for the IBM MQ service.

   The user account that IBM MQ Windows service runs under executes any MQSC commands that are issued by user interface applications, or performed automatically on system startup, shutdown, or service recovery. This user account must therefore have IBM MQ administration rights. By default it is added to the local **mqm** group on the server. If this membership is removed, the IBM MQ Windows service does not work. For more information about user rights, see “User rights required for an IBM MQ Windows Service” on page 65

   If a security problem arises with the user account that the IBM MQ Windows service runs under, error messages and descriptions appear in the system event log.

Related concepts:

"Using Active directory (Windows only)" on page 65
In some network configurations, where user accounts are defined on domain controllers that are using Active Directory, the local user account IBM MQ is running under might not have the authority it requires to query the group membership of other domain user accounts. The Prepare IBM MQ Wizard identifies whether this is the case by carrying out tests and asking the user questions about the network configuration.

IBM MQ coordinating with Db2 as the resource manager

If you start your queue managers from the MQ Explorer, or are using IBM MQ V7, and are having problems when coordinating Db2, check your queue manager error logs.

Check your queue manager error logs for an error like the following:

```
23/09/2008 15:43:54 - Process(5508.1) User(MUSR_MQADMIN) Program(amqzxma0.exe)
Host(HOST_1) Installation(Installation1)
VMRF(7.1.0.0) QMgr(A.B.C)
AMQ7604: The XA resource manager 'DB2 MQBankDB database' was not available when called
for xa_open. The queue manager is continuing without this resource manager.
```

Explanation: The user ID (default name is MUSR_MQADMIN) which runs the IBM MQ Service process `amqsvc.exe` is still running with an access token which does not contain group membership information for the group DB2USERS.

Solve: After you have ensured that the IBM MQ Service user ID is a member of DB2USERS, use the following sequence of commands:

- stop the service.
- stop any other processes running under the same user ID.
- restart these processes.

Rebooting the machine would ensure the previous steps, but is not necessary.

Extending the MQ Explorer

IBM MQ for Windows and IBM MQ for Linux x86-64 provide an administration interface called the MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands.

This information applies to IBM MQ for Windows, and IBM MQ for Linux x86-64 platforms only.

The MQ Explorer presents information in a style consistent with that of the Eclipse framework and the other plug-in applications that Eclipse supports.

Through extending the MQ Explorer, system administrators have the ability to customize the MQ Explorer to improve the way they administer IBM MQ.

For more information, see *Extending the MQ Explorer* in the MQ Explorer product documentation.

Using the IBM MQ Taskbar application (Windows only)

The IBM MQ Taskbar application displays an icon in the Windows system tray on the server. The icon provides you with the current status of IBM MQ and a menu from which you can perform some simple actions.

On Windows, the IBM MQ icon is in the system tray on the server and is overlaid with a color-coded status symbol, which can have one of the following meanings:

Green Working correctly; no alerts at present

Blue Indeterminate; IBM MQ is starting up or shutting down

Yellow
Alert; one or more services are failing or have already failed

To display the menu, right-click the IBM MQ icon. From the menu you can perform the following actions:

- Click **Open** to open the IBM MQ Alert Monitor
- Click **Exit** to exit the IBM MQ Taskbar application
- Click **IBM MQ Explorer** to start the IBM MQ Explorer
- Click **Stop IBM MQ** to stop IBM MQ
- Click **About IBM MQ** to display information about the IBM MQ Alert Monitor

The IBM MQ alert monitor application (Windows only)

The IBM MQ alert monitor is an error detection tool that identifies and records problems with IBM MQ on a local machine.

The alert monitor displays information about the current status of the local installation of an IBM MQ server. It also monitors the Windows Advanced Configuration and Power Interface (ACPI) and ensures the ACPI settings are enforced.

From the IBM MQ alert monitor, you can:

- Access the IBM MQ Explorer directly
- View information relating to all outstanding alerts
- Shut down the IBM MQ service on the local machine
- Route alert messages over the network to a configurable user account, or to a Windows workstation or server

Administering local IBM MQ objects

This section tells you how to administer local IBM MQ objects to support application programs that use the Message Queue Interface (MQI). In this context, local administration means creating, displaying, changing, copying, and deleting IBM MQ objects.

In addition to the approaches detailed in this section you can use the IBM MQ Explorer to administer local IBM MQ objects; see “Administration using the MQ Explorer” on page 56.

This section contains the following information:
- Application programs using the MQI
- “Performing local administration tasks using MQSC commands” on page 73
- “Working with queue managers” on page 82
- “Working with local queues” on page 84
- “Working with alias queues” on page 89
- “Working with model queues” on page 110
- “Working with services” on page 117
- “Managing objects for triggering” on page 124

Starting and stopping a queue manager

Use this topic as an introduction to stopping and starting a queue manager.

Starting a queue manager

To start a queue manager, use the **strmqm** command as follows:

```
strmqm saturn.queue.manager
```

On IBM MQ for Windows and IBM MQ for Linux (x86 and x86-64 platforms) systems, you can start a queue manager as follows:
1. Open the IBM MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click **Start** . The queue manager starts.

If the queue manager start-up takes more than a few seconds IBM MQ issues information messages intermittently detailing the start-up progress.

The **strmqm** command does not return control until the queue manager has started and is ready to accept connection requests.

Starting a queue manager automatically

In IBM MQ for Windows you can start a queue manager automatically when the system starts using the IBM MQ Explorer. For more information, see “Administration using the MQ Explorer” on page 56.

Stopping a queue manager

Use the **`endmqm`** command to stop a queue manager.

Note: You must use the **`endmqm`** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

For example, to stop a queue manager called QMB, enter the following command:

```
endmqm QMB
```

On IBM MQ for Windows and IBM MQ for Linux (x86 and x86-64 platforms) systems, you can stop a queue manager as follows:

1. Open the IBM MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click **Stop...** . The End Queue Manager panel is displayed.
4. Select Controlled, or Immediate.
5. Click **OK** . The queue manager stops.

Quiesced shutdown

By default, the **endmqm** command performs a quiesced shutdown of the specified queue manager. This might take a while to complete. A quiesced shutdown waits until all connected applications have disconnected.

Use this type of shutdown to notify applications to stop. If you issue:

```
endmqm -c QMB
```

you are not told when all applications have stopped. (An `endmqm -c QMB` command is equivalent to an `endmqm QMB` command.)

However, if you issue:

```
endmqm -w QMB
```

the command waits until all applications have stopped and the queue manager has ended.

Immediate shutdown

For an immediate shutdown any current MQI calls are allowed to complete, but any new calls fail. This type of shutdown does not wait for applications to disconnect from the queue manager.

For an immediate shutdown, type:

```
endmqm -i QMB
```

Preemptive shutdown

Note: Do not use this method unless all other attempts to stop the queue manager using the **endmqm** command have failed. This method can have unpredictable consequences for connected applications.

If an immediate shutdown does not work, you must resort to a *preemptive* shutdown, specifying the `-p` flag. For example:

```
endmqm -p QMB
```

This stops the queue manager immediately. If this method still does not work, see "Stopping a queue manager manually" on page 71 for an alternative solution.

For a detailed description of the **endmqm** command and its options, see endmqm.

If you have problems shutting down a queue manager

Problems in shutting down a queue manager are often caused by applications. For example, when applications:

- Do not check MQI return codes properly
- Do not request notification of a quiesce
- Terminate without disconnecting from the queue manager (by issuing an **MQDISC** call)

If a problem occurs when you stop the queue manager, you can break out of the **endmqm** command using Ctrl-C. You can then issue another **endmqm** command, but this time with a flag that specifies the type of shutdown that you require.

Stopping a queue manager manually

If the standard methods for stopping queue managers fail, try the methods described here.

The standard way of stopping queue managers is by using the **endmqm** command. To stop a queue manager manually, use one of the procedures described in this section. For details of how to perform operations on queue managers using control commands, see Creating and managing queue managers on distributed platforms.

Stopping queue managers in IBM MQ for Windows:

How to end the processes and the IBM MQ service, to stop queue managers in IBM MQ for Windows.

To stop a queue manager running under IBM MQ for Windows:

1. List the names (IDs) of the processes that are running, by using the Windows Task Manager.
2. End the processes by using Windows Task Manager, or the **taskkill** command, in the following order (if they are running):

AMQZMUC0	Critical process manager
AMQZXMA0	Execution controller
AMQZFUMA	OAM process
AMQZLAA0	LQM agents
AMQZLSA0	LQM agents
AMQZMUF0	Utility Manager
AMQZMGR0	Process controller
AMQZMUR0	Restartable process manager
AMQFQPUB	Publish Subscribe process
AMQFCXBA	Broker worker process
AMQRMPPA	Process pooling process
AMQCRSTA	Non-threaded responder job process
AMQCRS6B	LU62 receiver channel and client connection
AMQRRMFA	The repository process (for clusters)
AMQPCSEA	The command server
RUNMQTRM	Invoke a trigger monitor for a server
RUNMQDLQ	Invoke dead-letter queue handler
RUNMQCHI	The channel initiator process
RUNMQLSR	The channel listener process
AMQXSSVN	Shared memory servers

3. Stop the IBM MQ service from **Administration tools** > **Services** on the Windows Control Panel.
4. If you have tried all methods and the queue manager has not stopped, reboot your system.

The Windows Task Manager and the **tasklist** command give limited information about tasks. For more information to help to determine which processes relate to a particular queue manager, consider using a tool such as *Process Explorer* (procexp.exe), available for download from the Microsoft website at http://www.microsoft.com.

Stopping queue managers in IBM MQ for UNIX and Linux systems:

How to end the processes and the IBM MQ service, to stop queue managers in IBM MQ for UNIX and Linux. You can try the methods described here if the standard methods for stopping and removing queue managers fail.

To stop a queue manager running under IBM MQ for UNIX and Linux systems:

1. Find the process IDs of the queue manager programs that are still running by using the **ps** command. For example, if the queue manager is called `QMNAME`, use the following command:

   ```
   ps -ef | grep QMNAME
   ```

2. End any queue manager processes that are still running. Use the **kill** command, specifying the process IDs discovered by using the **ps** command.

   End the processes in the following order:

amqzmuc0	Critical process manager
amqzxma0	Execution controller
amqzfuma	OAM process
amqzlaa0	LQM agents
amqzlsa0	LQM agents
amqzmuf0	Utility Manager
amqzmur0	Restartable process manager
amqzmgr0	Process controller
amqfqpub	Publish Subscribe process
amqfcxba	Broker worker process
amqrmppa	Process pooling process
amqcrsta	Non-threaded responder job process
amqcrs6b	LU62 receiver channel and client connection
amqrrmfa	The repository process (for clusters)
amqpcsea	The command server
runmqtrm	Invoke a trigger monitor for a server
runmqdlq	Invoke dead-letter queue handler
runmqchi	The channel initiator process
runmqlsr	The channel listener process

Note: You can use the **`kill -9`** command to end processes that fail to stop.

If you stop the queue manager manually, FFST™ might be taken, and FDC files placed in `/var/mqm/errors`. Do not regard this as a defect in the queue manager.

The queue manager will restart normally, even after you have stopped it using this method.

Stopping MQI channels

When you issue a STOP CHANNEL command against a server-connection channel, you can choose what method to use to stop the client-connection channel. This means that a client channel issuing an MQGET wait call can be controlled, and you can decide how and when to stop the channel.

The STOP CHANNEL command can be issued with three modes, indicating how the channel is to be stopped:

Quiesce
Stops the channel after any current messages have been processed.

If sharing conversations is enabled, the IBM MQ MQI client becomes aware of the stop request in a timely manner; this time is dependent upon the speed of the network. The client application becomes aware of the stop request as a result of issuing a subsequent call to IBM MQ.

Force Stops the channel immediately.

Terminate
Stops the channel immediately. If the channel is running as a process, it can terminate the channel's process, or if the channel is running as a thread, its thread.

This is a multi-stage process. If mode terminate is used, an attempt is made to stop the server-connection channel, first with mode quiesce, then with mode force, and if necessary with mode terminate. The client can receive different return codes during the different stages of termination. If the process or thread is terminated, the client receives a communication error.

The return codes returned to the application vary according to the MQI call issued, and the STOP CHANNEL command issued. The client will receive either an MQRC_CONNECTION_QUIESCING or an MQRC_CONNECTION_BROKEN return code. If a client detects MQRC_CONNECTION_QUIESCING it should try to complete the current transaction and terminate. This is not possible with MQRC_CONNECTION_BROKEN. If the client does not complete the transaction and terminate fast enough it will get CONNECTION_BROKEN after a few seconds. A STOP CHANNEL command with MODE(FORCE) or MODE(TERMINATE) is more likely to result in a CONNECTION_BROKEN than with MODE(QUIESCE).

Related information:
MQI channels

Performing local administration tasks using MQSC commands

This section introduces you to MQSC commands and tells you how to use them for some common tasks.

If you use IBM MQ for Windows or IBM MQ for Linux (x86 and x86-64 platforms), you can also perform the operations described in this section using the IBM MQ Explorer. See “Administration using the MQ Explorer” on page 56 for more information.

You can use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects. This section deals with queue managers, queues, and process definitions; for an overview of channel, client connection channel, and listener objects, see Objects. For information about all the MQSC commands for managing queue manager objects, see “Script (MQSC) Commands” on page 74.

You issue MQSC commands to a queue manager using the **runmqsc** command. (For details of this command, see runmqsc.) You can do this interactively, issuing commands from a keyboard, or you can redirect the standard input device (`stdin`) to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same. (For information about running the commands from a text file, see “Running MQSC commands from text files” on page 78.)

You can run the **runmqsc** command in three ways, depending on the flags set on the command:

- Verify a command without running it, where the MQSC commands are verified on a local queue manager, but are not run.
- Run a command on a local queue manager, where the MQSC commands are run on a local queue manager.
- Run a command on a remote queue manager, where the MQSC commands are run on a remote queue manager.

You can also run the command followed by a question mark to display the syntax.

Object attributes specified in MQSC commands are shown in this section in uppercase (for example, RQMNAME), although they are not case-sensitive. MQSC command attribute names are limited to eight characters. MQSC commands are available on other platforms, including IBM i and z/OS.

From IBM MQ Version 8.0, you can set a prompt of your choice by using the MQPROMPT environment variable. In addition to plain text, the MQPROMPT variable also allows environment variables to be inserted, by using `+VARNAME+ notation`, in the same manner as IBM MQ service object definitions (see "Defining a service object" on page 118). For example:

```
sh> export MQPROMPT="+USER+ @ +QMNAME+ @ +MQ_HOST_NAME+> "
sh> runmqsc MY.QMGR
5724-H72 (C) Copyright IBM Corp. 1994, 2014.
Starting MQSC for queue manager MY.QMGR.
username @ MY.QMGR @ aix1> DISPLAY QMSTATUS
```

MQSC commands are detailed in the MQSC commands section.

Related information:

runmqsc (run MQSC commands)

Script (MQSC) Commands

MQSC commands provide a uniform method of issuing human-readable commands on IBM MQ platforms. For information about *programmable command format* (PCF) commands, see "Introduction to Programmable Command Formats" on page 6.

The general format of the commands is shown in The MQSC commands.

You should observe the following rules when using MQSC commands:

- Each command starts with a primary parameter (a verb), and this is followed by a secondary parameter (a noun). This is then followed by the name or generic name of the object (in parentheses) if there is one, which there is on most commands. Following that, parameters can usually occur in any order; if a parameter has a corresponding value, the value must occur directly after the parameter to which it relates.

  Note: z/OS On z/OS, the secondary parameter does not have to be second.
- Keywords, parentheses, and values can be separated by any number of blanks and commas. A comma shown in the syntax diagrams can always be replaced by one or more blanks. There must be at least one blank immediately preceding each parameter (after the primary parameter) z/OS except on z/OS .
- Any number of blanks can occur at the beginning or end of the command, and between parameters, punctuation, and values. For example, the following command is valid:

  ```
  ALTER QLOCAL ('Account' )      TRIGDPTH ( 1)
  ```

  Blanks within a pair of quotation marks are significant.

- Additional commas can appear anywhere where blanks are allowed and are treated as if they were blanks (unless, of course, they are inside strings enclosed by quotation marks).
- Repeated parameters are not allowed. Repeating a parameter with its "NO" version, as in REPLACE NOREPLACE, is also not allowed.
- Strings that contain blanks, lowercase characters or special characters other than:
  - Period (.)
  - Forward slash (/)
  - Underscore (_)
  - Percent sign (%)

  must be enclosed in single quotation marks, unless they are:
  - z/OS Issued from the IBM MQ for z/OS operations and control panels
  - Generic values ending with an asterisk IBM i (on IBM i these must be enclosed in single quotation marks)
  - A single asterisk (for example, TRACE(*)) IBM i (on IBM i these must be enclosed in single quotation marks)
  - A range specification containing a colon (for example, CLASS(01:03))

  If the string itself contains a single quotation mark, the single quotation mark is represented by two single quotation marks. Lowercase characters not contained within quotation marks are folded to uppercase.
- On platforms other than z/OS, a string containing no characters (that is, two single quotation marks with no space in between) is interpreted as a blank space enclosed in single quotation marks, that is, interpreted in the same way as (' '). The exception to this is if the attribute being used is one of the following:
  - `TOPICSTR`
  - `SUB`
  - `USERDATA`
  - `SELECTOR`

  then two single quotation marks with no space are interpreted as a zero-length string.

  z/OS On z/OS, if you want a blank space enclosed in single quotation marks, you must enter it as such (' '). A string containing no characters ('') is the same as entering ().
- In v7.0, any trailing blanks in those string attributes which are based on MQCHARV types, such as SELECTOR, sub user data, are treated as significant which means that 'abc ' does not equal 'abc'.
- A left parenthesis followed by a right parenthesis, with no significant information in between, for example

  ```
  NAME ( )
  ```

  is not valid except where specifically noted.
- Keywords are not case sensitive: AltER, alter, and ALTER are all acceptable. Anything that is not contained within quotation marks is folded to uppercase.
- Synonyms are defined for some parameters. For example, DEF is always a synonym for DEFINE, so DEF QLOCAL is valid. Synonyms are not, however, just minimum strings; DEFI is not a valid synonym for DEFINE.

  Note: There is no synonym for the DELETE parameter. This is to avoid accidental deletion of objects when using DEF, the synonym for DEFINE.

For an overview of using MQSC commands for administering IBM MQ, see “Performing local administration tasks using MQSC commands” on page 73.

MQSC commands use certain special characters to have certain meanings. For more information about these special characters and how to use them, see Characters with special meanings.

To find out how you can build scripts using MQSC commands, see Building command scripts.

z/OS For an explanation of the symbols in the z/OS column, see Using commands on z/OS.

For the full list of MQSC commands, see The MQSC commands.

Related information:

Building command scripts

IBM MQ object names

How to use object names in MQSC commands.

In examples, we use some long names for objects. This is to help you identify the type of object you are dealing with.

When you issue MQSC commands, you need specify only the local name of the queue. In our examples, we use queue names such as:

```
ORANGE.LOCAL.QUEUE
```

The `LOCAL.QUEUE` part of the name is to illustrate that this queue is a local queue. It is ***not*** required for the names of local queues in general.

We also use the name `saturn.queue.manager` as a queue manager name. The `queue.manager` part of the name is to illustrate that this object is a queue manager. It is *not* required for the names of queue managers in general.

Case-sensitivity in MQSC commands

MQSC commands, including their attributes, can be written in uppercase or lowercase. Object names in MQSC commands are folded to uppercase (that is, QUEUE and queue are not differentiated), unless the names are enclosed within single quotation marks. If quotation marks are not used, the object is processed with a name in uppercase. See Characters with special meanings for more information.

The **runmqsc** command invocation, in common with all IBM MQ control commands, is case sensitive in some IBM MQ environments. See Using control commands for more information.

Standard input and output

The *standard input device*, also referred to as `stdin`, is the device from which input to the system is taken. Typically this is the keyboard, but you can specify that input is to come from a serial port or a disk file, for example. The *standard output device*, also referred to as `stdout`, is the device to which output from the system is sent. Typically this is a display, but you can redirect output to a serial port or a file.

On operating-system commands and IBM MQ control commands, the < operator redirects input. If this operator is followed by a file name, input is taken from the file. Similarly, the > operator redirects output; if this operator is followed by a file name, output is directed to that file.

Using MQSC commands interactively

You can use MQSC commands interactively by using a command window or shell.

To use MQSC commands interactively, open a command window or shell and enter:

```
runmqsc
```

In this command, a queue manager name has not been specified, so the MQSC commands are processed by the default queue manager. If you want to use a different queue manager, specify the queue manager name on the **runmqsc** command. For example, to run MQSC commands on queue manager `jupiter.queue.manager`, use the command:

```
runmqsc jupiter.queue.manager
```

After this, all the MQSC commands you type in are processed by this queue manager, assuming that it is on the same node and is already running.

Now you can type in any MQSC commands, as required. For example, try this one:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE)
```

For commands that have too many parameters to fit on one line, use continuation characters to indicate that a command is continued on the following line:

- A minus sign (-) indicates that the command is to be continued from the start of the following line.
- A plus sign (+) indicates that the command is to be continued from the first nonblank character on the following line.

Command input terminates with the final character of a nonblank line that is not a continuation character. You can also terminate command input explicitly by entering a semicolon (;). (This is especially useful if you accidentally enter a continuation character at the end of the final line of command input.)

Feedback from MQSC commands

When you issue MQSC commands, the queue manager returns operator messages that confirm your actions or tell you about the errors you have made. For example:

```
AMQ8006: IBM MQ queue created.
```

This message confirms that a queue has been created.

```
AMQ8405: Syntax error detected at or near end of command segment below:-

AMQ8426: Valid MQSC commands are:


ALTER
CLEAR
DEFINE
DELETE
DISPLAY
END
PING
REFRESH
RESET
RESOLVE
RESUME
START
STOP
SUSPEND
4 : end
```

This message indicates that you have made a syntax error.

These messages are sent to the standard output device. If you have not entered the command correctly, refer to MQSC commands for the correct syntax.

Ending interactive input of MQSC commands

To stop working with MQSC commands, enter the END command.

Alternatively, you can use the EOF character for your operating system.

Running MQSC commands from text files

Running MQSC commands interactively is suitable for quick tests, but if you have very long commands, or are using a particular sequence of commands repeatedly, consider redirecting `stdin` from a text file.

"Standard input and output" on page 76 contains information about `stdin` and `stdout`. To redirect `stdin` from a text file, first create a text file containing the MQSC commands using your usual text editor. When you use the **runmqsc** command, use the redirection operators. For example, the following command runs a sequence of commands contained in the text file `myprog.in`:

```
runmqsc < myprog.in
```

Similarly, you can also redirect the output to a file. A file containing the MQSC commands for input is called an *MQSC command file*. The output file containing replies from the queue manager is called the *output file*.

To redirect both `stdin` and `stdout` on the **runmqsc** command, use this form of the command:

```
runmqsc < myprog.in > myprog.out
```

This command invokes the MQSC commands contained in the MQSC command file `myprog.in.` Because we have not specified a queue manager name, the MQSC commands run against the default queue manager. The output is sent to the text file `myprog.out`. Figure 11 on page 79 shows an extract from the MQSC command file `myprog.in` and Figure 12 on page 80 shows the corresponding extract of the output in `myprog.out`.

To redirect `stdin` and `stdout` on the **runmqsc** command, for a queue manager (`saturn.queue.manager`) that is not the default, use this form of the command:

```
runmqsc saturn.queue.manager < myprog.in > myprog.out
```

MQSC command files

MQSC commands are written in human-readable form, that is, in ASCII text. Figure 11 on page 79 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes. MQSC commands contains a description of each MQSC command and its syntax.

```
.
.
.
DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) REPLACE +
DESCR(' ') +
PUT(ENABLED) +
DEFPRTY(0) +
DEFPSIST(NO) +
GET(ENABLED) +
MAXDEPTH(5000) +
MAXMSGL(1024) +
DEFSOPT(SHARED) +
NOHARDENBO +
USAGE(NORMAL) +
NOTRIGGER;
.
.
.
```

Figure 11. Extract from an MQSC command file

For portability among IBM MQ environments, limit the line length in MQSC command files to 72 characters. The plus sign indicates that the command is continued on the next line.

MQSC command reports

The **runmqsc** command returns a *report*, which is sent to `stdout`. The report contains:

- A header identifying MQSC commands as the source of the report:

  ```
  Starting MQSC for queue manager jupiter.queue.manager.
  ```

  Where `jupiter.queue.manager` is the name of the queue manager.
- An optional numbered listing of the MQSC commands issued. By default, the text of the input is echoed to the output. Within this output, each command is prefixed by a sequence number, as shown in Figure 12 on page 80. However, you can use the -e flag on the **runmqsc** command to suppress the output.
- A syntax error message for any commands found to be in error.
- An *operator message* indicating the outcome of running each command. For example, the operator message for the successful completion of a DEFINE QLOCAL command is:

  ```
  AMQ8006: IBM MQ queue created.
  ```
- Other messages resulting from general errors when running the script file.
- A brief statistical summary of the report indicating the number of commands read, the number of commands with syntax errors, and the number of commands that could not be processed.

  Note: The queue manager attempts to process only those commands that have no syntax errors.

```
Starting MQSC for queue manager jupiter.queue.manager.
.
.
12:   DEFINE QLOCAL('ORANGE.LOCAL.QUEUE') REPLACE +
:       DESCR(' ') +
:       PUT(ENABLED) +
:       DEFPRTY(0) +
:       DEFPSIST(NO) +
:       GET(ENABLED) +
:       MAXDEPTH(5000) +
:       MAXMSGL(1024) +
:       DEFSOPT(SHARED) +
:       NOHARDENBO +
:       USAGE(NORMAL) +
:       NOTRIGGER;
AMQ8006: IBM MQ queue created.
:
.
.
```

Figure 12. Extract from an MQSC command report file

Running the supplied MQSC command files

The following MQSC command files are supplied with IBM MQ:

amqscos0.tst
Definitions of objects used by sample programs.

amqscic0.tst
Definitions of queues for CICS® transactions.

In IBM MQ for Windows, these files are located in the directory *MQ_INSTALLATION_PATH*\tools\mqsc\samples. *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.

On UNIX and Linux systems these files are located in the directory *MQ_INSTALLATION_PATH*/samp. *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.

The command that runs them is:

```
runmqsc < amqscos0.tst >test.out
```

Using runmqsc to verify commands

You can use the **runmqsc** command to verify MQSC commands on a local queue manager without actually running them. To do this, set the -v flag in the **runmqsc** command, for example:

```
runmqsc -v < myprog.in > myprog.out
```

When you invoke **runmqsc** against an MQSC command file, the queue manager verifies each command and returns a report without actually running the MQSC commands. This allows you to check the syntax of the commands in your command file. This is particularly important if you are:

- Running a large number of commands from a command file.
- Using an MQSC command file many times over.

The returned report is similar to that shown in Figure 12.

You cannot use this method to verify MQSC commands remotely. For example, if you attempt this command:

```
runmqsc -w 30 -v jupiter.queue.manager < myprog.in > myprog.out
```

the -w flag, which you use to indicate that the queue manager is remote, is ignored, and the command is run locally in verification mode. 30 is the number of seconds that IBM MQ waits for replies from the remote queue manager.

Running MQSC commands from batch files

If you have very long commands, or are using a particular sequence of commands repeatedly, consider redirecting `stdin` from a batch file.

To redirect `stdin` from a batch file, first create a batch file containing the MQSC commands using your usual text editor. When you use the **runmqsc** command, use the redirection operators. The following example:

1. Creates a test queue manager, TESTQM
2. Creates a matching CLNTCONN and listener set to use TCP/IP port 1600
3. Creates a test queue, TESTQ
4. Puts a message on the queue, using the amqsputc sample program

```
export MYTEMPQM=TESTQM
export MYPORT=1600
export MQCHLLIB=/var/mqm/qmgrs/$MQTEMPQM/@ipcc

crtmqm $MYTEMPQM
strmqm $MYTEMPQM
runmqlsr -m $MYTEMPQM -t TCP -p $MYPORT &

runmqsc $MYTEMPQM << EOF
DEFINE CHANNEL(NTLM) CHLTYPE(SVRCONN) TRPTYPE(TCP)
DEFINE CHANNEL(NTLM) CHLTYPE(CLNTCONN) QMNAME('$MYTEMPQM') CONNAME('hostname($MYPORT)')
ALTER CHANNEL(NTLM) CHLTYPE(CLNTCONN)
DEFINE QLOCAL(TESTQ)
EOF

amqsputc TESTQ $MYTEMPQM << EOF
hello world
EOF

endmqm -i $MYTEMPQM
```

Figure 13. Example script for running MQSC commands from a batch file

Resolving problems with MQSC commands

If you cannot get MQSC commands to run, use the information in this topic to see if any of these common problems apply to you. It is not always obvious what the problem is when you read the error that a command generates.

When you use the **runmqsc** command, remember the following:

- Use the < operator to redirect input from a file. If you omit this operator, the queue manager interprets the file name as a queue manager name, and issues the following error message:

  ```
  AMQ8118: IBM MQ queue manager does not exist.
  ```
- If you redirect output to a file, use the > redirection operator. By default, the file is put in the current working directory at the time **runmqsc** is invoked. Specify a fully-qualified file name to send your output to a specific file and directory.
- Check that you have created the queue manager that is going to run the commands, by using the following command to display all queue managers:

  ```
  dspmq
  ```
- The queue manager must be running. If it is not, start it; (see Starting a queue manager). You get an error message if you try to start a queue manager that is already running.

- Specify a queue manager name on the **runmqsc** command if you have not defined a default queue manager, or you get this error:

```
AMQ8146: IBM MQ queue manager not available.
```

- You cannot specify an MQSC command as a parameter of the **runmqsc** command. For example, this is not valid:

```
runmqsc DEFINE QLOCAL(FRED)
```

- You cannot enter MQSC commands before you issue the **runmqsc** command.
- You cannot run control commands from **runmqsc** . For example, you cannot issue the **strmqm** command to start a queue manager while you are running MQSC commands interactively. If you do this, you receive error messages similar to the following:

```
runmqsc
.
.
Starting MQSC for queue manager jupiter.queue.manager.

1 : strmqm saturn.queue.manager
AMQ8405: Syntax error detected at or near end of cmd segment below:-s

AMQ8426: Valid MQSC commands are:
ALTER
CLEAR
DEFINE
DELETE
DISPLAY
END
PING
REFRESH
RESET
RESOLVE
RESUME
START
STOP
SUSPEND
2 : end
```

Working with queue managers

Examples of MQSC commands that you can use to display or alter queue manager attributes.

Displaying queue manager attributes

To display the attributes of the queue manager specified on the **runmqsc** command, use the following MQSC command:

```
DISPLAY QMGR
```

Typical output from this command is shown in Figure 14 on page 83

```
DISPLAY QMGR
     1 : DISPLAY QMGR
AMQ8408: Display Queue Manager details.
   QMNAME(QM1)                            ACCTCONO(DISABLED)
   ACCTINT(1800)                          ACCTMQI(OFF)
   ACCTQ(OFF)                             ACTIVREC(MSG)
   ACTVCONO (DISABLED)                    ACTVTRC (OFF)
   ALTDATE(2012-05-27)                    ALTTIME(16.14.01)
   AUTHOREV(DISABLED)                     CCSID(850)
   CHAD(DISABLED)                         CHADEV(DISABLED)
   CHADEXIT( )                            CHLEV(DISABLED)
   CLWLDATA( )                            CLWLEXIT( )
   CLWLLEN(100)                           CLWLMRUC(999999999)
   CLWLUSEQ(LOCAL)                        CMDEV(DISABLED)
   CMDLEVEL(800)                          COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
   CONFIGEV(DISABLED)                     CRDATE(2011-05-27)
   CRTIME(16.14.01)                       DEADQ()
   DEFXMITQ( )                            DESCR( )
   DISTL(YES)                             INHIBTEV(DISABLED)
   IPADDRV(IPV4)                          LOCALEV(DISABLED)
   LOGGEREV(DISABLED)                     MARKINT(5000)
   MAXHANDS(256)                          MAXMSGL(4194304)
   MAXPROPL(NOLIMIT)                      MAXPRTY(9)
   MAXUMSGS(10000)                        MONACLS(QMGR)
   MONCHL(OFF)                            MONQ(OFF)
   PARENT( )                              PERFMEV(DISABLED)
   PLATFORM(WINDOWSNT)                    PSRTYCNT(5)
   PSNPMSG(DISCARD)                       PSNPRES(NORMAL)
   PSSYNCPT(IFPER)                        QMID(QM1_2011-05-27_16.14.01)
   PSMODE(ENABLED)                        REMOTEEV(DISABLED)
   REPOS( )                               REPOSNL( )
   ROUTEREC(MSG)                          SCHINIT(QMGR)
   SCMDSERV(QMGR)                         SSLCRLNL( )
   SSLCRYP( )                             SSLEV(DISABLED)
   SSLFIPS(NO)                            SSLKEYR(C:\Program Files\IBM\WebSphere
MQ\Data\qmgrs\QM1\ssl\key)
   SSLRKEYC(0)                            STATACLS(QMGR)
   STATCHL(OFF)                           STATINT(1800)
   STATMQI(OFF)                           STATQ(OFF)
   STRSTPEV(ENABLED)                      SYNCPT
   TREELIFE(1800)                         TRIGINT(999999999)
```

Figure 14. Typical output from a DISPLAY QMGR command

The `ALL` parameter is the default on the `DISPLAY QMGR` command. It displays all the queue manager attributes. In particular, the output tells you the default queue manager name, the dead-letter queue name, and the command queue name.

You can confirm that these queues exist by entering the command:

```
DISPLAY QUEUE (SYSTEM.*)
```

This displays a list of queues that match the stem `SYSTEM.*`. The parentheses are required.

Altering queue manager attributes

To alter the attributes of the queue manager specified on the **runmqsc** command, use the MQSC command `ALTER QMGR`, specifying the attributes and values that you want to change. For example, use the following commands to alter the attributes of `jupiter.queue.manager`:

```
runmqsc jupiter.queue.manager
ALTER QMGR DEADQ (ANOTHERDLQ) INHIBTEV (ENABLED)
```

The `ALTER QMGR` command changes the dead-letter queue used, and enables inhibit events.

Related information:

Attributes for the queue manager

Working with local queues

This section contains examples of some MQSC commands that you can use to manage local, model, and alias queues.

See MQSC commands for detailed information about these commands.

Defining a local queue

For an application, the local queue manager is the queue manager to which the application is connected. Queues managed by the local queue manager are said to be local to that queue manager.

Use the MQSC command DEFINE QLOCAL to create a local queue. You can also use the default defined in the default local queue definition, or you can modify the queue characteristics from those of the default local queue.

Note: The default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE and it was created on system installation.

For example, the DEFINE QLOCAL command that follows defines a queue called ORANGE.LOCAL.QUEUE with these characteristics:

- It is enabled for gets, enabled for puts, and operates on a priority order basis.
- It is an *normal* queue; it is not an initiation queue or transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 5000 messages; the maximum message length is 4194304 bytes.

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) +
DESCR('Queue for messages from other systems') +
PUT (ENABLED) +
GET (ENABLED) +
NOTRIGGER +
MSGDLVSQ (PRIORITY) +
MAXDEPTH (5000) +
MAXMSGL (4194304) +
USAGE (NORMAL);
```

Note:

1. With the exception of the value for the description, all the attribute values shown are the default values. We have shown them here for purposes of illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also "Displaying default object attributes" on page 85.
2. USAGE (NORMAL) indicates that this queue is not a transmission queue.
3. If you already have a local queue on the same queue manager with the name ORANGE.LOCAL.QUEUE, this command fails. Use the REPLACE attribute if you want to overwrite the existing definition of a queue, but see also "Changing local queue attributes" on page 86.

Displaying default object attributes

You can use the DISPLAY QUEUE command to display attributes that were taken from the default object when an IBM MQ object was defined.

When you define an IBM MQ object, it takes any attributes that you do not specify from the default object. For example, when you define a local queue, the queue inherits any attributes that you omit in the definition from the default local queue, which is called SYSTEM.DEFAULT.LOCAL.QUEUE. To see exactly what these attributes are, use the following command:

```
DISPLAY QUEUE (SYSTEM.DEFAULT.LOCAL.QUEUE)
```

The syntax of this command is different from that of the corresponding DEFINE command. On the DISPLAY command you can give just the queue name, whereas on the DEFINE command you have to specify the type of the queue, that is, QLOCAL, QALIAS, QMODEL, or QREMOTE.

You can selectively display attributes by specifying them individually. For example:

```
DISPLAY QUEUE (ORANGE.LOCAL.QUEUE) +
MAXDEPTH +
MAXMSGL +
CURDEPTH;
```

This command displays the three specified attributes as follows:

```
AMQ8409: Display Queue details.
QUEUE(ORANGE.LOCAL.QUEUE)       TYPE(QLOCAL)
CURDEPTH(0)              MAXDEPTH(5000)
MAXMSGL(4194304)
```

CURDEPTH is the current queue depth, that is, the number of messages on the queue. This is a useful attribute to display, because by monitoring the queue depth you can ensure that the queue does not become full.

Copying a local queue definition

You can copy a queue definition using the LIKE attribute on the DEFINE command.

For example:

```
DEFINE QLOCAL (MAGENTA.QUEUE) +
LIKE (ORANGE.LOCAL.QUEUE)
```

This command creates a queue with the same attributes as our original queue ORANGE.LOCAL.QUEUE, rather than those of the system default local queue. Enter the name of the queue to be copied ***exactly*** as it was entered when you created the queue. If the name contains lower case characters, enclose the name in single quotation marks.

You can also use this form of the DEFINE command to copy a queue definition, but substitute one or more changes to the attributes of the original. For example:

```
DEFINE QLOCAL (THIRD.QUEUE) +
LIKE (ORANGE.LOCAL.QUEUE) +
MAXMSGL(1024);
```

This command copies the attributes of the queue ORANGE.LOCAL.QUEUE to the queue THIRD.QUEUE, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 4194304.

Note:

1. When you use the LIKE attribute on a DEFINE command, you are copying the queue attributes only. You are not copying the messages on the queue.

2. If you a define a local queue, without specifying LIKE, it is the same as DEFINE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE).

Changing local queue attributes

You can change queue attributes in two ways, using either the ALTER QLOCAL command or the DEFINE QLOCAL command with the REPLACE attribute.

In "Defining a local queue" on page 84, the queue called ORANGE.LOCAL.QUEUE was defined. Suppose, for example, that you want to decrease the maximum message length on this queue to 10,000 bytes.

- Using the ALTER command:

  ```
  ALTER QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000)
  ```

  This command changes a single attribute, that of the maximum message length; all the other attributes remain the same.
- Using the DEFINE command with the REPLACE option, for example:

  ```
  DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000) REPLACE
  ```

  This command changes not only the maximum message length, but also all the other attributes, which are given their default values. The queue is now put enabled whereas previously it was put inhibited. Put enabled is the default, as specified by the queue SYSTEM.DEFAULT.LOCAL.QUEUE.

  If you ***decrease*** the maximum message length on an existing queue, existing messages are not affected. Any new messages, however, must meet the new criteria.

Clearing a local queue

You can use the CLEAR command to clear a local queue.

To delete all the messages from a local queue called MAGENTA.QUEUE, use the following command:

```
CLEAR QLOCAL (MAGENTA.QUEUE)
```

Note: There is no prompt that enables you to change your mind; once you press the Enter key the messages are lost.

You cannot clear a queue if:

- There are uncommitted messages that have been put on the queue under sync point.
- An application currently has the queue open.

Deleting a local queue

You can use the MQSC command DELETE QLOCAL to delete a local queue.

A queue cannot be deleted if it has uncommitted messages on it. However, if the queue has one or more committed messages and no uncommitted messages, it can be deleted only if you specify the PURGE option. For example:

```
DELETE QLOCAL (PINK.QUEUE) PURGE
```

Specifying NOPURGE instead of PURGE ensures that the queue is not deleted if it contains any committed messages.

Browsing queues

IBM MQ provides a sample queue browser that you can use to look at the contents of the messages on a queue. The browser is supplied in both source and executable formats.

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

In IBM MQ for Windows, the file names and paths for the sample queue browser are as follows:

Source
: `MQ_INSTALLATION_PATH\tools\c\samples\`

Executable
: `MQ_INSTALLATION_PATH\tools\c\samples\bin\amqsbcg.exe`

In IBM MQ for UNIX and Linux, the file names and paths are as follows:

Source
: `MQ_INSTALLATION_PATH/samp/amqsbcg0.c`

Executable
: `MQ_INSTALLATION_PATH/samp/bin/amqsbcg`

The sample requires two input parameters, the queue name and the queue manager name. For example:

```
amqsbcg SYSTEM.ADMIN.QMGREVENT.tpp01 saturn.queue.manager
```

Typical results from this command are shown in Figure 15 on page 88.

```
AMQSBCG0 - starts here
**********************

 MQOPEN - 'SYSTEM.ADMIN.QMGR.EVENT'


 MQGET of message number 1
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 850
  Format : 'MQEVENT '
  Priority : 0  Persistence : 0
  MsgId : X'414D512073617475726E2E71756575650005D30033563DB8'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'saturn.queue.manager                            '
  ** Identity Context
  UserIdentifier : '            '
  AccountingToken :
   X'0000000000000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '7'
  PutApplName    : 'saturn.queue.manager        '
  PutDate  : '19970417'    PutTime  : '15115208'
  ApplOriginData : '    '

  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '104'

****  Message   ****

length - 104 bytes

00000000: 0700 0000 2400 0000 0100 0000 2C00 0000 '....→.......,...'
00000010: 0100 0000 0100 0000 0100 0000 AE08 0000 '................'
00000020: 0100 0000 0400 0000 4400 0000 DF07 0000 '........D.......'
00000030: 0000 0000 3000 0000 7361 7475 726E 2E71 '....0...saturn.q'
00000040: 7565 7565 2E6D 616E 6167 6572 2020 2020 'ueue.manager    '
00000050: 2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060: 2020 2020 2020 2020                     '                '

No more messages
MQCLOSE
MQDISC
```

Figure 15. Typical results from queue browser

Related information:

The Browser sample program

Enabling large queues

IBM MQ supports queues larger than 2 GB.

On Windows systems, support for large files is available without any additional enablement. On AIX, HP-UX, Linux, and Solaris systems, you need to explicitly enable large file support before you can create queue files larger than 2 GB. See your operating system documentation for information on how to do this.

Some utilities, such as tar, cannot cope with files greater than 2 GB. Before enabling large file support, check your operating system documentation for information on restrictions on utilities you use.

For information about planning the amount of storage you need for queues, visit the IBM MQ website for platform-specific performance reports: `http://www.ibm.com/support/docview.wss?rs=171&uid=swg27007150&loc=en_US&cs=utf-8&lang=en#1`

Working with alias queues

You can define an alias queue to refer indirectly to another queue or topic.

V 8.0.0.6

Attention: Distribution lists do not support the use of alias queues that point to topic objects. From Version 8.0.0, Fix Pack 6, if an alias queue points to a topic object in a distribution list, IBM MQ returns MQRC_ALIAS_BASE_Q_TYPE_ERROR.

The queue to which an alias queue refers can be any of the following:
- A local queue (see "Defining a local queue" on page 84).
- A local definition of a remote queue (see "Creating a local definition of a remote queue" on page 137).
- A topic.

An alias queue is not a real queue, but a definition that resolves to a real (or target) queue at run time. The alias queue definition specifies the target queue. When an application makes an **MQOPEN** call to an alias queue, the queue manager resolves the alias to the target queue name.

An alias queue cannot resolve to another locally defined alias queue. However, an alias queue can resolve to alias queues that are defined elsewhere in clusters of which the local queue manager is a member. See Name resolution for further information.

Alias queues are useful for:
- Giving different applications different levels of access authorities to the target queue.
- Allowing different applications to work with the same queue in different ways. (Perhaps you want to assign different default priorities or different default persistence values.)
- Simplifying maintenance, migration, and workload balancing. (Perhaps you want to change the target queue name without having to change your application, which continues to use the alias.)

For example, assume that an application has been developed to put messages on a queue called MY.ALIAS.QUEUE. It specifies the name of this queue when it makes an **MQOPEN** request and, indirectly, if it puts a message on this queue. The application is not aware that the queue is an alias queue. For each MQI call using this alias, the queue manager resolves the real queue name, which could be either a local queue or a remote queue defined at this queue manager.

By changing the value of the TARGQ attribute, you can redirect MQI calls to another queue, possibly on another queue manager. This is useful for maintenance, migration, and load-balancing.

Defining an alias queue

The following command creates an alias queue:

```
DEFINE QALIAS (MY.ALIAS.QUEUE) TARGET (YELLOW.QUEUE)
```

This command redirects MQI calls that specify MY.ALIAS.QUEUE to the queue YELLOW.QUEUE. The command does not create the target queue; the MQI calls fail if the queue YELLOW.QUEUE does not exist at run time.

If you change the alias definition, you can redirect the MQI calls to another queue. For example:

```
ALTER QALIAS (MY.ALIAS.QUEUE) TARGET (MAGENTA.QUEUE)
```

This command redirects MQI calls to another queue, MAGENTA.QUEUE.

You can also use alias queues to make a single queue (the target queue) appear to have different attributes for different applications. You do this by defining two aliases, one for each application. Suppose there are two applications:

- Application ALPHA can put messages on YELLOW.QUEUE, but is not allowed to get messages from it.
- Application BETA can get messages from YELLOW.QUEUE, but is not allowed to put messages on it.

The following command defines an alias that is put enabled and get disabled for application ALPHA:

```
DEFINE QALIAS (ALPHAS.ALIAS.QUEUE) +
TARGET (YELLOW.QUEUE) +
PUT (ENABLED) +
GET (DISABLED)
```

The following command defines an alias that is put disabled and get enabled for application BETA:

```
DEFINE QALIAS (BETAS.ALIAS.QUEUE) +
TARGET (YELLOW.QUEUE) +
PUT (DISABLED) +
GET (ENABLED)
```

ALPHA uses the queue name ALPHAS.ALIAS.QUEUE in its MQI calls; BETA uses the queue name BETAS.ALIAS.QUEUE. They both access the same queue, but in different ways.

You can use the LIKE and REPLACE attributes when you define queue aliases, in the same way that you use these attributes with local queues.

Using other commands with alias queues

You can use the appropriate MQSC commands to display or alter alias queue attributes, or to delete the alias queue object. For example:

Use the following command to display the alias queue's attributes:

```
DISPLAY QUEUE (ALPHAS.ALIAS.QUEUE)
```

Use the following command to alter the base queue name, to which the alias resolves, where the `force` option forces the change even if the queue is open:

```
ALTER QALIAS (ALPHAS.ALIAS.QUEUE) TARGQ(ORANGE.LOCAL.QUEUE) FORCE
```

Use the following command to delete this queue alias:

```
DELETE QALIAS (ALPHAS.ALIAS.QUEUE)
```

You cannot delete an alias queue if an application currently has the queue open. See MQSC commands for more information about this, and other alias queue commands.

Working with dead-letter queues

Each queue manager typically has a local queue to use as a dead-letter queue, so that messages that cannot be delivered to their correct destination can be stored for later retrieval. You tell the queue manager about the dead-letter queue, and specify how messages found on a dead-letter queue are to be processed. Using dead-letter queues can affect the sequence in which messages are delivered, so you might choose not to use them.

To tell the queue manager about the dead-letter queue, specify a dead-letter queue name on the **crtmqm** command (`crtmqm -u DEAD.LETTER.QUEUE`, for example), or by using the DEADQ attribute on the ALTER QMGR command to specify one later. You must define the dead-letter queue before using it.

A sample dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE is available with the product. This queue is automatically created when you create the queue manager. You can modify this definition if required, and rename it.

A dead-letter queue has no special requirements except that:
- It must be a local queue
- Its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle **plus** the size of the dead-letter header (MQDLH)

Using dead-letter queues can affect the sequence in which messages are delivered, so you might choose not to use them. You set the USEDLQ channel attribute to determine whether the dead-letter queue is used when messages cannot be delivered. This attribute can be configured so that some functions of the queue manager use the dead-letter queue, while other functions do not. For more information about the use of the USEDLQ channel attribute on different MQSC commands, see DEFINE CHANNEL, DISPLAY CHANNEL, ALTER CHANNEL, and DISPLAY CLUSQMGR.

IBM MQ provides a dead-letter queue handler that allows you to specify how messages found on a dead-letter queue are to be processed or removed. See "Processing messages on a dead-letter queue."

Related information:

Dead-letter queues

Undelivered messages troubleshooting

Processing messages on a dead-letter queue

To process messages on a dead-letter queue (DLQ), MQ supplies a default DLQ handler. The handler matches messages on the DLQ against entries in a rules table that you define.

Messages can be put on a DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ must be prefixed with a *dead-letter header* structure, MQDLH. Messages put on the DLQ by a queue manager or a message channel agent always have this header; applications putting messages on the DLQ must supply this header. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

All IBM MQ environments need a routine to process messages on the DLQ regularly. IBM MQ supplies a default routine, called the *dead-letter queue handler* (the DLQ handler), which you invoke using the **runmqdlq** command.

Instructions for processing messages on the DLQ are supplied to the DLQ handler by means of a user-written *rules table*. That is, the DLQ handler matches messages on the DLQ against entries in the rules table; when a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

Related information:
Dead-letter queues
Undelivered messages troubleshooting

The IBM MQ for IBM i dead-letter queue handler:

Use this information to learn about, and how to invoke, the dead-letter queue handler.

A *dead-letter queue* (DLQ), sometimes referred to as an *undelivered-message queue*, is a holding queue for messages that cannot be delivered to their destination queues. Every queue manager in a network should have an associated DLQ.

Note: It is often preferable to avoid placing messages on a DLQ. For information about the use and avoidance of DLQs, see "Working with dead-letter queues" on page 91.

Queue managers, message channel agents, and applications can put messages on the DLQ. All messages on the DLQ must be prefixed with a *dead-letter header* structure, MQDLH. Messages put on the DLQ by a queue manager or by a message channel agent always have an MQDLH. Always supply an MQDLH to applications putting messages on the DLQ. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

In all IBM MQ environments, there must be a routine that runs regularly to process messages on the DLQ. IBM MQ supplies a default routine, called the *dead-letter queue handler* (the DLQ handler), which you invoke using the STRMQMDLQ command. A user-written *rules table* supplies instructions to the DLQ handler, for processing messages on the DLQ. That is, the DLQ handler matches messages on the DLQ against entries in the rules table. When a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

Invoking the DLQ handler

Use the STRMQMDLQ command to invoke the DLQ handler. You can name the DLQ that you want to process and the queue manager that you want to use in two ways:

- As parameters to STRMQMDLQ from the command prompt. For example:
  ```
  STRMQMDLQ UDLMSGQ(ABC1.DEAD.LETTER.QUEUE) SRCMBR(QRULE) SRCFILE(library/QTXTSRC)
  MQMNAME(MY.QUEUE.MANAGER)
  ```
- In the rules table. For example:
  ```
  INPUTQ(ABC1.DEAD.LETTER.QUEUE)
  ```

Note: The rules table is a member within a source physical file that can take any name.

The examples apply to the DLQ called `ABC1.DEAD.LETTER.QUEUE`, owned by the default queue manager.

If you do not specify the DLQ or the queue manager as shown, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The STRMQMDLQ command takes its input from the rules table.

You must be authorized to access both the DLQ itself, and any message queues to which messages on the DLQ are forwarded, in order to run the DLQ handler. You must also be authorized to assume the identity of other users, for the DLQ to put messages on queues with the authority of the user ID in the message context.

Related information:
Dead-letter queues
Undelivered messages troubleshooting

The DLQ handler rules table:

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ. There are two types of entry in a rules table:
- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

Control data

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table. Note the following:
- The default value for a keyword, if any, is underlined.
- The vertical line (|) separates alternatives. You can specify only one of these.
- All keywords are optional.

INPUTQ (*QueueName* **|' ')**
The name of the DLQ you want to process:
1. Any UDLMSGQ value (or *DFT) you specify as a parameter to the **STRMQMDLQ** command overrides any INPUTQ value in the rules table.
2. If you specify a blank UDLMSGQ value as a parameter to the **STRMQMDLQ** command, the INPUTQ value in the rules table is used.
3. If you specify a blank UDLMSGQ value as a parameter to the **STRMQMDLQ** command, and a blank INPUTQ value in the rules table, the system default dead-letter queue is used.

INPUTQM (*QueueManagerName* **|' ')**
The name of the queue manager that owns the DLQ named on the INPUTQ keyword.

If you do not specify a queue manager, or you specify INPUTQM(' ') in the rules table, the system uses the default queue manager for the installation.

RETRYINT (*Interval* **|** <u>**60**</u> **)**
The interval, in seconds, at which the DLQ handler should attempt to reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. By default, the retry interval is 60 seconds.

WAIT (<u>**YES**</u> **|NO|** *nnn* **)**
Whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES Causes the DLQ handler to wait indefinitely.

NO Causes the DLQ handler to terminate when it detects that the DLQ is either empty or contains no messages that it can process.

nnn Causes the DLQ handler to wait for *nnn* seconds for new work to arrive before terminating, after it detects that the queue is either empty or contains no messages that it can process.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, re-invoke it using triggering.

You can supply the name of the DLQ as an input parameter to the **STRMQMDLQ** command, as an alternative to including control data in the rules table. If any value is specified both in the rules table and on input to the **STRMQMDLQ** command, the value specified on the **STRMQMDLQ** command takes precedence.

Note: If a control-data entry is included in the rules table, it ***must*** be the first entry in the table.

Rules (patterns and actions):

Use this information to understand the DLQ rules.

Here is an example rule from a DLQ handler rules table:

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
ACTION (RETRY) RETRY (3)
```

This rule instructs the DLQ handler to make 3 attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

This section describes the keywords that you can include in a rule. Note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives. You can specify only one of these.
- All keywords except ACTION are optional.

This section begins with a description of the pattern-matching keywords (those against which messages on the DLQ are matched). It then describes the action keywords (those that determine how the DLQ handler is to process a matching message).

The pattern-matching keywords:

The pattern-matching keywords are described in the following example. Use them to specify values against which messages on the DLQ are matched. All pattern-matching keywords are optional.

APPLIDAT (*ApplIdentityData* **|** **<u>*</u>** **)**
The *ApplIdentityData* value of the message on the DLQ, specified in the message descriptor, MQMD.

APPLNAME (*PutApplName* **|** **<u>*</u>** **)**
The name of the application that issued the MQPUT or MQPUT1 call, as specified in the *PutApplName* field of the message descriptor, MQMD, of the message on the DLQ.

APPLTYPE (*PutApplType* **|** **<u>*</u>** **)**
The *PutApplType* value specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName* **|** **<u>*</u>** **)**
The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName* **|** **<u>*</u>** **)**
The queue manager name for the message queue for which the message is destined.

FEEDBACK (*Feedback* **|** **<u>*</u>** **)**
When the *MsgType* value is MQMT_REPORT, *Feedback* describes the nature of the report.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that require confirmation of their arrival on their destination queues.

FORMAT (*Format* **|** **<u>*</u>** **)**
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType* **|** ***** **)**
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that require replies.

PERSIST (*Persistence* **|** ***** **)**
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify those messages on the DLQ that are persistent.

REASON (*ReasonCode* **|** ***** **)**
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName* **|** ***** **)**
The reply-to queue name specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName* **|** ***** **)**
The queue manager name of the reply-to queue specified in the REPLYQ keyword.

USERID (*UserIdentifier* **|** ***** **)**
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD.

The action keywords:

The action keywords are described. Use them to determine how a matching message is processed.

ACTION (DISCARD|IGNORE|RETRY|FWD)
The action taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Causes the message to be deleted from the DLQ.

IGNORE
Causes the message to be left on the DLQ.

RETRY
Causes the DLQ handler to try again to put the message on its destination queue.

FWD Causes the DLQ handler to forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword. The number of attempts made to implement an action is governed by the RETRY keyword. The RETRYINT keyword of the control data controls the interval between attempts.

FWDQ (*QueueName* **|&DESTQ|&REPLYQ)**
The name of the message queue to which the message is forwarded when you select the ACTION keyword.

QueueName
The name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Take the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Take the queue name from the *ReplyToQ* field in the message descriptor, MQMD.

You can specify REPLYQ (?*) in the message pattern to avoid error messages, when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field.

FWDQM (*QueueManagerName* **|&DESTQM|&REPLYQM|' ')**
The queue manager of the queue to which a message is forwarded.

QueueManagerName
The queue manager name for the queue to which the message is forwarded when you select the ACTION (FWD) keyword.

&DESTQM
Take the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Take the queue manager name from the *ReplyToQMgr* field in the message descriptor, MQMD.

' ' FWDQM(' '), which is the default value, identifies the local queue manager.

HEADER (**YES** **|NO)**
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (**DEF** **|CTX)**
The authority with which messages should be put by the DLQ handler:

DEF Puts messages with the authority of the DLQ handler itself.

CTX Causes the messages to be put with the authority of the user ID in the message context. You must be authorized to assume the identity of other users, if you specify PUTAUT (CTX).

RETRY (*RetryCount* **|** **1** **)**
The number of times, in the range 1 - 999,999,999, to attempt an action (at the interval specified on the RETRYINT keyword of the control data).

Note: The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If you restart the DLQ handler, the count of attempts made to apply a rule is reset to zero.

Rules table conventions:

The rules table must adhere to the following conventions regarding its syntax, structure, and contents.

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included once only in any rule.
- Keywords are not case sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- Any number of blanks can occur at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- For portability, the significant length of a line must not be greater than 72 characters.
- Use the plus sign (+) as the last non-blank character on a line to indicate that the rule continues from the first non-blank character in the next line. Use the minus sign (-) as the last non-blank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:

```
APPLNAME('ABC+
D')
```

results in 'ABCD'.

```
APPLNAME('ABC-
D')
```

results in 'ABC D'.

- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting quotation marks in values enclosed in quotation marks are not considered significant. For example, these parameters are valid:

`FORMAT('ABC')`	3 significant characters
`FORMAT(ABC)`	3 significant characters
`FORMAT('A')`	1 significant character
`FORMAT(A)`	1 significant character
`FORMAT(' ')`	1 significant character

These parameters are invalid because they contain no significant characters:

```
FORMAT('')
FORMAT( )
FORMAT()
FORMAT
```

  - Wildcard characters are supported. You can use the question mark (?) in place of any single character, except a trailing blank. You can use the asterisk (*) in place of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.
  - You cannot include wildcard characters in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
  - Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.
  - Numeric parameters cannot include the question mark (?) wildcard character. You can include the asterisk (*) in place of an entire numeric parameter, but the asterisk cannot be included as part of a numeric parameter. For example, these are valid numeric parameters:

`MSGTYPE(2)`	Only reply messages are eligible
`MSGTYPE(*)`	Any message type is eligible
`MSGTYPE('*')`	Any message type is eligible

However, `MSGTYPE('2*')` is not valid, because it includes an asterisk (*) as part of a numeric parameter.

  - Numeric parameters must be in the range 0-999 999 999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.
  - If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8-character

field:

`'ABCDEFGH'`	8 characters
`'A*C*E*G*I'`	5 characters excluding asterisks
`'*A*C*E*G*I*K*M*O*'`	8 characters excluding asterisks

– Strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (?), underscore (_), and percent sign (%) must be enclosed in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation mark, two single quotation marks must be used to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

Processing the rules table:

The DLQ handler searches the rules table for a rule with a pattern that matches a message on the DLQ.

The search begins with the first rule in the table, and continues sequentially through the table. When a rule with a matching pattern is found, the rules table attempts the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it attempts to apply that rule. If the first attempt fails, the attempt is repeated until the count of attempts made matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

Note:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can default, so that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors flagged at that time. (Error messages issued by the DLQ handler are described in Reason codes.) You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler is restarted.
4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized, because the validation of the rules table eliminates the generation of repetitive errors.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed:

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed.

If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still keeps a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ will be seen, even if the DLQ is defined as first-in first-out (FIFO). If the queue is not empty, the DLQ is periodically re-scanned to check all messages.

For these reasons, try to ensure that the DLQ contains as few messages as possible. If messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself is in danger of filling up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, try not to use ACTION (IGNORE), which leaves messages on the DLQ. (Remember that ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table.) Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, make the final rule in the table a catchall to process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be something like this:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This causes messages that fall through to the final rule in the table to be forwarded to the queue `REALLY.DEAD.QUEUE`, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

An example DLQ handler rules table:

Here is an example rules table that contains a single control-data entry and several rules:

```
************************************************************************
*    An example rules table for the STRMQMDLQ command       *
************************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* STRMQMDLQ, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to STRMQMDLQ,
* use the DLQ defined for the local queue manager.
*
inputqm(' ') inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.

* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation is always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never does things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it must be able
* to cope with the message being lost, so we can afford to
* discard the message.

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We do not have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

Invoking the DLQ handler:

Invoke the DLQ handler using the **runmqdlq** command. You can name the DLQ you want to process and the queue manager you want to use in two ways.

The two ways are as follows:
- As parameters to **runmqdlq** from the command prompt. For example:
  ```
  runmqdlq ABC1.DEAD.LETTER.QUEUE ABC1.QUEUE.MANAGER <qrule.rul
  ```
- In the rules table. For example:
  ```
  INPUTQ(ABC1.DEAD.LETTER.QUEUE) INPUTQM(ABC1.QUEUE.MANAGER)
  ```

The examples apply to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the queue manager ABC1.QUEUE.MANAGER.

If you do not specify the DLQ or the queue manager as shown, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The **runmqdlq** command takes its input from `stdin` ; you associate the rules table with **runmqdlq** by redirecting `stdin` from the rules table.

To run the DLQ handler you must be authorized to access both the DLQ itself and any message queues to which messages on the DLQ are forwarded. For the DLQ handler to put messages on queues with the authority of the user ID in the message context, you must also be authorized to assume the identity of other users.

For more information about the **runmqdlq** command, see runmqdlq.
Related information:
Dead-letter queues
Undelivered messages troubleshooting

The sample DLQ handler, amqsdlq:

In addition to the DLQ handler invoked using the **runmqdlq** command, IBM MQ provides the source of a sample DLQ handler, amqsdlq, with a function that is similar to that provided by **runmqdlq** .

You can customize amqsdlq to provide a DLQ handler that meets your requirements. For example, you might decide that you want a DLQ handler that can process messages without dead-letter headers. (Both the default DLQ handler and the sample, amqsdlq, process only those messages on the DLQ that begin with a dead-letter header, MQDLH. Messages that do not begin with an MQDLH are identified as being in error, and remain on the DLQ indefinitely.)

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

In IBM MQ for Windows, the source of amqsdlq is supplied in the directory:
```
MQ_INSTALLATION_PATH\tools\c\samples\dlq
```

and the compiled version is supplied in the directory:
```
MQ_INSTALLATION_PATH\tools\c\samples\bin
```

In IBM MQ for UNIX and Linux systems, the source of amqsdlq is supplied in the directory:

```
MQ_INSTALLATION_PATH/samp/dlq
```

and the compiled version is supplied in the directory:

```
MQ_INSTALLATION_PATH/samp/bin
```

The DLQ handler rules table:

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ.

There are two types of entry in a rules table:
- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.
Related information:
Dead-letter queues
Undelivered messages troubleshooting

Control data:

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table.

Note:
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords are optional.

INPUTQ (*QueueName* **|' ')**
The name of the DLQ you want to process:
1. Any INPUTQ value you supply as a parameter to the **runmqdlq** command overrides any INPUTQ value in the rules table.
2. If you do not specify an INPUTQ value as a parameter to the **runmqdlq** command, but you **do** specify a value in the rules table, the INPUTQ value in the rules table is used.
3. If no DLQ is specified or you specify INPUTQ(' ') in the rules table, the name of the DLQ belonging to the queue manager with the name that is supplied as a parameter to the **runmqdlq** command is used.
4. If you do not specify an INPUTQ value as a parameter to the **runmqdlq** command or as a value in the rules table, the DLQ belonging to the queue manager named on the INPUTQM keyword in the rules table is used.

INPUTQM (*QueueManagerName* **|' ')**
The name of the queue manager that owns the DLQ named on the INPUTQ keyword:
1. Any INPUTQM value you supply as a parameter to the **runmqdlq** command overrides any INPUTQM value in the rules table.
2. If you do not specify an INPUTQM value as a parameter to the **runmqdlq** command, the INPUTQM value in the rules table is used.
3. If no queue manager is specified or you specify INPUTQM(' ') in the rules table, the default queue manager for the installation is used.

RETRYINT (*Interval* **|** **<u>60</u>** **)**
The interval, in seconds, at which the DLQ handler should reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. By default, the retry interval is 60 seconds.

WAIT (YES |NO| *nnn*)
Whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES The DLQ handler waits indefinitely.

NO The DLQ handler ends when it detects that the DLQ is either empty or contains no messages that it can process.

nnn The DLQ handler waits for *nnn* seconds for new work to arrive before ending, after it detects that the queue is either empty or contains no messages that it can process.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, invoke it again using triggering. For more information about triggering, see Starting IBM MQ applications using triggers.

An alternative to including control data in the rules table is to supply the names of the DLQ and its queue manager as input parameters to the **runmqdlq** command. If you specify a value both in the rules table and as input to the **runmqdlq** command, the value specified on the **runmqdlq** command takes precedence.

If you include a control-data entry in the rules table, it must be the **first** entry in the table.

Rules (patterns and actions):

A description of the pattern-matching keywords (those against which messages on the DLQ are matched), and the action keywords (those that determine how the DLQ handler is to process a matching message). An example rule is also provided.

The pattern-matching keywords

The pattern-matching keywords, which you use to specify values against which messages on the DLQ are matched, are as follows. (All pattern-matching keywords are optional):

APPLIDAT (*ApplIdentityData* | *)
The *ApplIdentityData* value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (*PutApplName* | *)
The name of the application that issued the **MQPUT** or **MQPUT1** call, as specified in the *PutApplName* field of the message descriptor, MQMD, of the message on the DLQ.

APPLTYPE (*PutApplType* | *)
The *PutApplType* value, specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName* | *)
The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName* | *)
The name of the queue manager of the message queue for which the message is destined.

FEEDBACK (*Feedback* | *)
When the *MsgType* value is MQFB_REPORT, *Feedback* describes the nature of the report.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that need confirmation of their arrival on their destination queues.

FORMAT (*Format* | *)
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType* | *)
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that need replies.

PERSIST (*Persistence* **|** <u>*</u> **)**
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify messages on the DLQ that are persistent.

REASON (*ReasonCode* **|** <u>*</u> **)**
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName* **|** <u>*</u> **)**
The name of the reply-to queue specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName* **|** <u>*</u> **)**
The name of the queue manager of the reply-to queue, as specified in the message descriptor, MQMD, of the message on the DLQ.

USERID (*UserIdentifier* **|** <u>*</u> **)**
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD, of the message on the DLQ.

The action keywords

The action keywords, used to describe how a matching message is to be processed, are as follows:

ACTION (DISCARD|IGNORE|RETRY|FWD)
The action to be taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Delete the message from the DLQ.

IGNORE
Leave the message on the DLQ.

RETRY
If the first attempt to put the message on its destination queue fails, try again. The RETRY keyword sets the number of tries made to implement an action. The RETRYINT keyword of the control data controls the interval between attempts.

FWD Forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword.

FWDQ (*QueueName* **|&DESTQ|&REPLYQ)**
The name of the message queue to which to forward the message when ACTION (FWD) is requested.

QueueName
The name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Take the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Take the queue name from the *ReplyToQ* field in the message descriptor, MQMD.

To avoid error messages when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field, specify REPLYQ (?*) in the message pattern.

FWDQM (*QueueManagerName* **|&DESTQM|&REPLYQM|' ')**
The queue manager of the queue to which to forward a message.

QueueManagerName
The name of the queue manager of the queue to which to forward a message when ACTION (FWD) is requested.

&DESTQM
Take the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Take the queue manager name from the *ReplyToQMgr* field in the message descriptor, MQMD.

' ' FWDQM(' '), which is the default value, identifies the local queue manager.

HEADER (**YES** **|NO)**
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (**DEF** **|CTX)**
The authority with which messages should be put by the DLQ handler:

DEF Put messages with the authority of the DLQ handler itself.

CTX Put the messages with the authority of the user ID in the message context. If you specify PUTAUT (CTX), you must be authorized to assume the identity of other users.

RETRY (*RetryCount* **|** **1** **)**
The number of times, in the range 1 - 999,999,999, to try an action (at the interval specified on the RETRYINT keyword of the control data). The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If the DLQ handler is restarted, the count of attempts made to apply a rule is reset to zero.

Example rule

Here is an example rule from a DLQ handler rules table:

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
ACTION (RETRY) RETRY (3)
```

This rule instructs the DLQ handler to make three attempts to deliver to its destination queue any persistent message that was put on the DLQ because **MQPUT** and **MQPUT1** were inhibited.

All keywords that you can use on a rule are described in the rest of this section. Note the following:
- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords except ACTION are optional.

Rules table conventions:

The syntax, structure and contents of the DLQ handler rules table must adhere to these conventions.

The rules table must adhere to the following conventions:
- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included only once in any rule.
- Keywords are not case-sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- There can be any number of blanks at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- On Windows systems, the last rule in the table must end with a carriage return/line feed character. You can achieve this by ensuring that you press the Enter key at the end of the rule, so that the last line of the table is a blank line.
- For reasons of portability, the significant length of a line must not be greater than 72 characters.
- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (-) as the last nonblank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

  For example:
  ```
  APPLNAME('ABC+
  D')
  ```
  results in 'ABCD', and
  ```
  APPLNAME('ABC-
  D')
  ```
  results in 'ABC D'.
- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting single quotation marks in values that are enclosed in quotation marks are not considered to be significant. For example, these parameters are valid:

`FORMAT('ABC')`	3 significant characters
`FORMAT(ABC)`	3 significant characters
`FORMAT('A')`	1 significant character
`FORMAT(A)`	1 significant character
`FORMAT(' ')`	1 significant character

    These parameters are invalid because they contain no significant characters:

```
FORMAT('')
FORMAT( )
FORMAT()
FORMAT
```

- Wildcard characters are supported. You can use the question mark (?) instead of any single character, except a trailing blank; you can use the asterisk (*) instead of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.
- Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
- Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings that are enclosed in single quotation marks are significant to wildcard matches.
- Numeric parameters cannot include the question mark (?) wildcard character. You can use the asterisk (*) instead of an entire numeric parameter, but not as part of a numeric parameter. For example, these are valid numeric parameters:

`MSGTYPE(2)`	Only reply messages are eligible
`MSGTYPE(*)`	Any message type is eligible
`MSGTYPE('*')`	Any message type is eligible

  However, `MSGTYPE('2*')` is not valid, because it includes an asterisk (*) as part of a numeric parameter.
- Numeric parameters must be in the range 0-999 999 999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.
- If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8 character field:

`'ABCDEFGH'`	8 characters
`'A*C*E*G*I'`	5 characters excluding asterisks
`'*A*C*E*G*I*K*M*O*'`	8 characters excluding asterisks

- Enclose strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (?), underscore (_), and percent sign (%) in single quotation marks. Lowercase characters not enclosed in single quotation marks are folded to uppercase. If the string includes a quotation, use two single quotation marks to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

How the rules table is processed:

The DLQ handler searches the rules table for a rule where the pattern matches a message on the DLQ.

The search begins with the first rule in the table, and continues sequentially through the table. When the DLQ handler finds a rule with a matching pattern, it takes the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it applies that rule. If the first try fails, the DLQ handler tries again until the number of tries matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

Note:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors are flagged at that time. You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler restarts.
4. The DLQ handler does not alter the content of messages, the MQDLH, or the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized because the rules table is designed to eliminate the generation of repetitive errors during validation.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Related information:

Dead-letter queues

Undelivered messages troubleshooting

Ensuring that all DLQ messages are processed:

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed.

If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still has to keep a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ are seen, even if the DLQ is defined as first-in-first-out (FIFO). If the queue is not empty, the DLQ is periodically re-scanned to check all messages.

For these reasons, try to ensure that the DLQ contains as few messages as possible; if messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself can fill up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, try not to use ACTION (IGNORE), which leaves messages on the DLQ. (Remember that ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table.) Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, make the final rule in the table a catchall to process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be something like this:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This forwards messages that fall through to the final rule in the table to the queue `REALLY.DEAD.QUEUE`, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

An example DLQ handler rules table:

An example rules table for the runmqdlq command, containing a single control-data entry and several rules.

```
*************************************************************************
*    An example rules table for the runmqdlq command       *
*************************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
inputqm(' ') inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.
* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation are always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never do things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)
```

```
* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it should be able
* to cope with the message being lost, so we can afford to
* discard the message.  PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)
* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We don't have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

Related information:

Dead-letter queues

Undelivered messages troubleshooting

Working with model queues

A queue manager creates a *dynamic queue* if it receives an MQI call from an application specifying a queue name that has been defined as a model queue. The name of the new dynamic queue is generated by the queue manager when the queue is created. A *model queue* is a template that specifies the attributes of any dynamic queues created from it. Model queues provide a convenient method for applications to create queues as required.

Defining a model queue

You define a model queue with a set of attributes in the same way that you define a local queue. Model queues and local queues have the same set of attributes, except that on model queues you can specify whether the dynamic queues created are temporary or permanent. (Permanent queues are maintained across queue manager restarts, temporary ones are not.) For example:

```
DEFINE QMODEL (GREEN.MODEL.QUEUE) +
DESCR('Queue for messages from application X') +
PUT (DISABLED) +
GET (ENABLED) +
NOTRIGGER +
MSGDLVSQ (FIFO) +
MAXDEPTH (1000) +
MAXMSGL (2000) +
USAGE (NORMAL) +
DEFTYPE (PERMDYN)
```

This command creates a model queue definition. From the DEFTYPE attribute, you can see that the actual queues created from this template are permanent dynamic queues. Any attributes not specified are automatically copied from the SYSYTEM.DEFAULT.MODEL.QUEUE default queue.

You can use the LIKE and REPLACE attributes when you define model queues, in the same way that you use them with local queues.

Using other commands with model queues

You can use the appropriate MQSC commands to display or alter a model queue's attributes, or to delete the model queue object. For example:

Use the following command to display the model queue's attributes:

```
DISPLAY QUEUE (GREEN.MODEL.QUEUE)
```

Use the following command to alter the model to enable puts on any dynamic queue created from this model:

```
ALTER QMODEL (BLUE.MODEL.QUEUE) PUT(ENABLED)
```

Use the following command to delete this model queue:

```
DELETE QMODEL (RED.MODEL.QUEUE)
```

Working with administrative topics

Use MQSC commands to manage administrative topics.

See MQSC commands for detailed information about these commands.

Related concepts:

"Defining an administrative topic"
Use the MQSC command **DEFINE TOPIC** to create an administrative topic. When defining an administrative topic you can optionally set each topic attribute.

"Displaying administrative topic object attributes" on page 112
Use the MQSC command **DISPLAY TOPIC** to display an administrative topic object.

"Changing administrative topic attributes" on page 113
You can change topic attributes in two ways, using either the **ALTER TOPIC** command or the **DEFINE TOPIC** command with the **REPLACE** attribute.

"Copying an administrative topic definition" on page 113
You can copy a topic definition using the LIKE attribute on the **DEFINE** command.

"Deleting an administrative topic definition" on page 114
You can use the MQSC command **DELETE TOPIC** to delete an administrative topic.

Related information:

Administrative topic objects

Defining an administrative topic

Use the MQSC command **DEFINE TOPIC** to create an administrative topic. When defining an administrative topic you can optionally set each topic attribute.

Any attribute of the topic that is not explicitly set is inherited from the default administrative topic, SYSTEM.DEFAULT.TOPIC, that was created when the system installation was installed.

For example, the **DEFINE TOPIC** command that follows, defines a topic called **ORANGE.TOPIC** with these characteristics:

- Resolves to the topic string ORANGE. For information about how topic strings can be used, see Combining topic strings.

- Any attribute that is set to ASPARENT uses the attribute as defined by the parent topic of this topic. This action is repeated up the topic tree as far as the root topic, SYSTEM.BASE.TOPIC is found. For more information, see Topic trees.

```
DEFINE TOPIC (ORANGE.TOPIC) +
TOPICSTR (ORANGE) +
DEFPRTY(ASPARENT) +
NPMSGDLV(ASPARENT)
```

Note:

- Except for the value of the topic string, all the attribute values shown are the default values. They are shown here only as an illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also "Displaying administrative topic object attributes."
- If you already have an administrative topic on the same queue manager with the name ORANGE.TOPIC, this command fails. Use the REPLACE attribute if you want to overwrite the existing definition of a topic, but see also "Changing administrative topic attributes" on page 113

Displaying administrative topic object attributes

Use the MQSC command **DISPLAY TOPIC** to display an administrative topic object.

To display all topics, use:

```
DISPLAY TOPIC(ORANGE.TOPIC)
```

You can selectively display attributes by specifying them individually. For example:

```
DISPLAY TOPIC(ORANGE.TOPIC) +
TOPICSTR +
DEFPRTY +
NPMSGDLV
```

This command displays the three specified attributes as follows:

```
AMQ8633: Display topic details.
     TOPIC(ORANGE.TOPIC)                                    TYPE(LOCAL)
     TOPICSTR(ORANGE)                                          DEFPRTY(ASPARENT)
     NPMSGDLV(ASPARENT)
```

To display the topic ASPARENT values as they are used at Runtime use DISPLAY TPSTATUS. For example, use:

```
DISPLAY TPSTATUS(ORANGE) DEFPRTY NPMSGDLV
```

The command displays the following details:

```
AMQ8754: Display topic status details.
TOPICSTR(ORANGE)           DEFPRTY(0)
NPMSGDLV(ALLAVAIL)
```

When you define an administrative topic, it takes any attributes that you do not specify explicitly from the default administrative topic, which is called SYSTEM.DEFAULT.TOPIC. To see what these default attributes are, use the following command:

```
DISPLAY TOPIC (SYSTEM.DEFAULT.TOPIC)
```

Changing administrative topic attributes

You can change topic attributes in two ways, using either the **ALTER TOPIC** command or the **DEFINE TOPIC** command with the **REPLACE** attribute.

If, for example, you want to change the default priority of messages delivered to a topic called ORANGE.TOPIC, to be 5, use either of the following commands.

- Using the **ALTER** command:

  ```
  ALTER TOPIC(ORANGE.TOPIC) DEFPRTY(5)
  ```

  This command changes a single attribute, that of the default priority of message delivered to this topic to 5; all other attributes remain the same.
- Using the **DEFINE** command:

  ```
  DEFINE TOPIC(ORANGE.TOPIC) DEFPRTY(5) REPLACE
  ```

  This command changes the default priority of messages delivered to this topic. All the other attributes are given their default values.

If you alter the priority of messages sent to this topic, existing messages are not affected. Any new message, however, use the specified priority if not provided by the publishing application.

Copying an administrative topic definition

You can copy a topic definition using the LIKE attribute on the **DEFINE** command.

For example:

```
DEFINE TOPIC (MAGENTA.TOPIC) +
LIKE (ORANGE.TOPIC)
```

This command creates a topic, MAGENTA.TOPIC, with the same attributes as the original topic, ORANGE.TOPIC, rather than those of the system default administrative topic. Enter the name of the topic to be copied exactly as it was entered when you created the topic. If the name contains lowercase characters, enclose the name in single quotation marks.

You can also use this form of the **DEFINE** command to copy a topic definition, but make changes to the attributes of the original. For example:

```
DEFINE TOPIC(BLUE.TOPIC) +
TOPICSTR(BLUE) +
LIKE(ORANGE.TOPIC)
```

You can also copy the attributes of the topic BLUE.TOPIC to the topic GREEN.TOPIC and specify that when publications cannot be delivered to their correct subscriber queue they are not placed onto the dead-letter queue. For example:

```
DEFINE TOPIC(GREEN.TOPIC) +
TOPICSTR(GREEN) +
LIKE(BLUE.TOPIC) +
USEDLQ(NO)
```

Deleting an administrative topic definition

You can use the MQSC command **DELETE TOPIC** to delete an administrative topic.

```
DELETE TOPIC(ORANGE.TOPIC)
```

Applications will no longer be able to open the topic for publication or make new subscriptions using the object name, ORANGE.TOPIC. Publishing applications that have the topic open are able to continue publishing the resolved topic string. Any subscriptions already made to this topic continue receiving publications after the topic has been deleted.

Applications that are not referencing this topic object but are using the resolved topic string that this topic object represented, 'ORANGE' in this example, continue to work. In this case they inherit the properties from a topic object higher in the topic tree. For more information, see Topic trees.

Working with subscriptions

Use MQSC commands to manage subscriptions.

Subscriptions can be one of three types, defined in the **SUBTYPE** attribute:

ADMIN
Administratively defined by a user.

PROXY
An internally created subscription for routing publications between queue managers.

API Created programmatically, for example, using the MQI MQSUB call.

See MQSC commands for detailed information about these commands.

Related concepts:

"Defining an administrative subscription"
Use the MQSC command **DEFINE SUB** to create an administrative subscription. You can also use the default defined in the default local subscription definition. Or, you can modify the subscription characteristics from those of the default local subscription, SYSTEM.DEFAULT.SUB that was created when the system was installed.

"Displaying attributes of subscriptions" on page 115
You can use the **DISPLAY SUB** command to display configured attributes of any subscription known to the queue manager.

"Changing local subscription attributes" on page 116
You can change subscription attributes in two ways, using either the **ALTER SUB** command or the **DEFINE SUB** command with the **REPLACE** attribute.

"Copying a local subscription definition" on page 116
You can copy a subscription definition using the **LIKE** attribute on the **DEFINE** command.

"Deleting a subscription" on page 117
You can use the MQSC command **DELETE SUB** to delete a local subscription.

Defining an administrative subscription

Use the MQSC command **DEFINE SUB** to create an administrative subscription. You can also use the default defined in the default local subscription definition. Or, you can modify the subscription characteristics from those of the default local subscription, SYSTEM.DEFAULT.SUB that was created when the system was installed.

For example, the **DEFINE SUB** command that follows defines a subscription called ORANGE with these characteristics:

- Durable subscription, meaning that it persists over queue manager restart, with unlimited expiry.
- Receive publications made on the ORANGE topic string, with the message priorities as set by the publishing applications.

- Publications delivered for this subscription are sent to the local queue SUBQ, this queue must be defined before the definition of the subscription.

```
DEFINE SUB (ORANGE) +
TOPICSTR (ORANGE) +
DESTCLAS (PROVIDED) +
DEST (SUBQ) +
EXPIRY (UNLIMITED) +
PUBPRTY (ASPUB)
```

Note:

- The subscription and topic string name do not have to match.
- Except for the values of the destination and topic string, all the attribute values shown are the default values. They are shown here only as an illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also “Displaying attributes of subscriptions.”
- If you already have a local subscription on the same queue manager with the name ORANGE, this command fails. Use the **REPLACE** attribute if you want to overwrite the existing definition of a queue, but see also “Changing local subscription attributes” on page 116.
- If the queue SUBQ does not exist, this command fails.

Displaying attributes of subscriptions

You can use the **DISPLAY SUB** command to display configured attributes of any subscription known to the queue manager.

For example, use:

```
DISPLAY SUB (ORANGE)
```

You can selectively display attributes by specifying them individually. For example:

```
DISPLAY SUB (ORANGE) +
SUBID +
TOPICSTR +
DURABLE
```

This command displays the three specified attributes as follows:

```
AMQ8096: IBM MQ subscription inquired.
     SUBID(414D5120414141202020202020202020EE921E4E20002A03)
     SUB(ORANGE)                                                          TOPICSTR(ORANGE)
     DURABLE(YES)
```

TOPICSTR is the resolved topic string on which this subscriber is operating. When a subscription is defined to use a topic object the topic string from that object is used as a prefix to the topic string provided when making the subscription. SUBID is a unique identifier assigned by the queue manager when a subscription is created. This is a useful attribute to display because some subscription names might be long or in a different character sets for which it might become impractical.

An alternate method for displaying subscriptions is to use the SUBID:

```
DISPLAY SUB +
SUBID(414D5120414141202020202020202020EE921E4E20002A03) +
TOPICSTR +
DURABLE
```

This command gives the same output as before:

```
AMQ8096: IBM MQ subscription inquired.
SUBID(414D5120414141202020202020202020EE921E4E20002A03)
     SUB(ORANGE)                                                          TOPICSTR(ORANGE)
     DURABLE(YES)
```

Proxy subscriptions on a queue manager are not displayed by default. To display them specify a **SUBTYPE** of PROXY or ALL.

You can use the DISPLAY SBSTATUS command to display the Runtime attributes. For example, use the command:

```
DISPLAY SBSTATUS(ORANGE) NUMMSGS
```

The following output is displayed:

```
AMQ8099: IBM MQ subscription status inquired.
SUB(ORANGE)
SUBID(414D5120414141202020202020202020EE921E4E20002A03)
NUMMSGS(0)
```

When you define an administrative subscription, it takes any attributes that you do not specify explicitly from the default subscription, which is called SYSTEM.DEFAULT.SUB. To see what these default attributes are, use the following command:

```
DISPLAY SUB (SYSTEM.DEFAULT.SUB)
```

Changing local subscription attributes

You can change subscription attributes in two ways, using either the **ALTER SUB** command or the **DEFINE SUB** command with the **REPLACE** attribute.

If, for example, you want to change the priority of messages delivered to a subscription called ORANGE to be 5, use either of the following commands:

- Using the ALTER command:
  ```
  ALTER SUB(ORANGE) PUBPRTY(5)
  ```
  This command changes a single attribute, that of the priority of messages delivered to this subscription to 5; all other attributes remain the same.
- Using the DEFINE command:
  ```
  DEFINE SUB (ORANGE) PUBPRTY(5) REPLACE
  ```
  This command changes not only the priority of messages delivered to this subscription, but all the other attributes which are given their default values.

If you alter the priority of messages sent to this subscription, existing messages are not affected. Any new messages, however, are of the specified priority.

Copying a local subscription definition

You can copy a subscription definition using the **LIKE** attribute on the **DEFINE** command.

For example:

```
DEFINE SUB (BLUE) +
LIKE (ORANGE)
```

You can also copy the attributes of the sub REAL to the sub THIRD.SUB, and specify that the correlID of delivered publications is THIRD, rather than the publishers correlID. For example:

```
DEFINE SUB(THIRD.SUB) +
LIKE(BLUE) +
DESTCORL(ORANGE)
```

Deleting a subscription

You can use the MQSC command **DELETE SUB** to delete a local subscription.

```
DELETE SUB(ORANGE)
```

You can also delete a subscription using the SUBID:

```
DELETE SUB SUBID(414D5120414141202020202020202020EE921E4E20002A03)
```

Checking messages on a subscription

About this task

When a subscription is defined it is associated with a queue. Published messages matching this subscription are put to this queue.

Note that the following **runmqsc** commands show only those subscriptions that received messages.

To check for messages currently queued for a subscription perform the following steps:

Procedure

1. To check for messages queued for a subscription type **DISPLAY SBSTATUS(<sub_name>) NUMMSGS**, see "Displaying attributes of subscriptions" on page 115.
2. If the **NUMMSGS** value is greater than zero identify the queue associated with the subscription by typing **DISPLAY SUB(<sub_name>)DEST**.
3. Using the name of the queue returned you can view the messages by following the technique described in "Browsing queues" on page 87.

Working with services

Service objects are a means by which additional processes can be managed as part of a queue manager. With services, you can define programs that are started and stopped when the queue manager starts and ends. IBM MQ services are always started under the user ID of the user who started the queue manager.

Service objects can be either of the following types:

Server A server is a service object that has the parameter SERVTYPE specified as SERVER. A server service object is the definition of a program that is executed when a specified queue manager is started. Server service objects define programs that typically run for a long time. For example, a server service object can be used to execute a trigger monitor process, such as **runmqtrm** .

Only one instance of a server service object can run concurrently. The status of running server service objects can be monitored using the MQSC command, DISPLAY SVSTATUS.

Command

A command is a service object that has the parameter SERVTYPE specified as COMMAND. Command service objects are similar to server service objects, however multiple instances of a command service object can run concurrently, and their status cannot be monitored using the MQSC command DISPLAY SVSTATUS.

If the MQSC command, STOP SERVICE, is executed no check is made to determine whether the program started by the MQSC command, START SERVICE, is still active before executing the stop program.

Defining a service object

You define a service object with various attributes.

The attributes are as follows:

SERVTYPE
Defines the type of the service object. Possible values are as follows:

SERVER
A server service object.

Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the MQSC command, DISPLAY SVSTATUS.

COMMAND
A command service object.

Multiple instances of a command service object can be executed concurrently. The status of a command service objects cannot be monitored.

STARTCMD
The program that is executed to start the service. A fully qualified path to the program must be specified.

STARTARG
Arguments passed to the start program.

STDERR
Specifies the path to a file to which the standard error (stderr) of the service program should be redirected.

STDOUT
Specifies the path to a file to which the standard output (stdout) of the service program should be redirected.

STOPCMD
The program that is executed to stop the service. A fully qualified path to the program must be specified.

STOPARG
Arguments passed to the stop program.

CONTROL
Specifies how the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is controlled by use of the START SERVICE and STOP SERVICE commands. This is the default value.

QMGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

Related concepts:

"Managing services"
By using the CONTROL parameter, an instance of a service object can be either started and stopped automatically by the queue manager, or started and stopped using the MQSC commands START SERVICE and STOP SERVICE.

Managing services

By using the CONTROL parameter, an instance of a service object can be either started and stopped automatically by the queue manager, or started and stopped using the MQSC commands START SERVICE and STOP SERVICE.

When an instance of a service object is started, a message is written to the queue manager error log containing the name of the service object and the process ID of the started process. An example log entry for a server service object starting follows:

```
02/15/2005 11:54:24 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5028: The Server 'S1' has started. ProcessId(13031).

EXPLANATION:
The Server process has started.
ACTION:
None.
```

An example log entry for a command service object starting follows:

```
02/15/2005 11:53:55 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5030: The Command 'C1' has started. ProcessId(13030).

EXPLANATION:
The Command has started.
ACTION:
None.
```

When an instance server service stops, a message is written to the queue manager error logs containing the name of the service and the process ID of the ending process. An example log entry for a server service object stopping follows:

```
02/15/2005 11:54:54 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr(A.B.C)
AMQ5029: The Server 'S1' has ended. ProcessId(13031).

EXPLANATION:
The Server process has ended.
ACTION:
None.
```

Related reference:

"Additional environment variables"
When a service is started, the environment in which the service process is started is inherited from the environment of the queue manager. It is possible to define additional environment variables to be set in the environment of the service process by adding the variables you want to define to one of the `service.env` environment override files.

Additional environment variables

When a service is started, the environment in which the service process is started is inherited from the environment of the queue manager. It is possible to define additional environment variables to be set in the environment of the service process by adding the variables you want to define to one of the `service.env` environment override files.

Note:

There are two possible files to which you can add environment variables:

- The machine scope `service.env` file, which is located in `/var/mqm` on UNIX and Linux systems, or in the data directory selected during installation on Windows systems.
- The queue manager scope `service.env` file, which is located in the queue manager data directory. For example, the location of the environment override file for a queue manager named `QMNAME` is:
  - On UNIX and Linux systems, `/var/mqm/qmgrs/QMNAME/service.env`
  - On Windows systems, `C:\ProgramData\IBM\MQ\qmgrs\QMNAME\service.env`

Both files are processed, if available, with definitions in the queue manager scope file taking precedence over those definitions in the machine scope file.

Any environment variable can be specified in `service.env`. For example, if the IBM MQ service runs a number of commands, it might be useful to set the PATH user variable in the `service.env` file. The values that you set the variable to can't be environment variables; for example CLASSPATH= *%CLASSPATH%* is incorrect. Similarly, on Linux PATH= *$PATH* `:/opt/mqm/bin` would give unexpected results.

CLASSPATH must be capitalized, and the class path statement can contain only literals. Some services (Telemetry for example) set their own class path. The CLASSPATH defined in `service.env` is added to it.

The format of the variables defined in the file, `service.env` is a list of name and value variable pairs. Each variable must be defined on a new line, and each variable is taken as it is explicitly defined, including white space. An example of the file, `service.env` follows:

```
#********************************************************************#
#*                                                                  *#
#* <N_OCO_COPYRIGHT>                                                *#
#* Licensed Materials - Property of IBM                             *#
#*                                                                  *#
#* 63H9336                                                          *#
#* (C) Copyright IBM Corporation 2005                               *#
#*                                                                  *#
#* <NOC_COPYRIGHT>                                                  *#
#*                                                                  *#
#********************************************************************#
#***********************************************************************#
#* Module Name: service.env                                            *#
#* Type       : IBM MQ service environment file                        *#
#* Function   : Define additional environment variables to be set      *#
#*              for SERVICE programs.                                  *#
#* Usage      : <VARIABLE>=<VALUE>                                     *#
#*                                                                     *#
#***********************************************************************#
```

```
MYLOC=/opt/myloc/bin
MYTMP=/tmp
TRACEDIR=/tmp/trace
MYINITQ=ACCOUNTS.INITIATION.QUEUE
```

Related reference:

"Replaceable inserts on service definitions"
In the definition of a service object, it is possible to substitute tokens. Tokens that are substituted are automatically replaced with their expanded text when the service program is executed. Substitute tokens can be taken from the following list of common tokens, or from any variables that are defined in the file, service.env.

Replaceable inserts on service definitions

In the definition of a service object, it is possible to substitute tokens. Tokens that are substituted are automatically replaced with their expanded text when the service program is executed. Substitute tokens can be taken from the following list of common tokens, or from any variables that are defined in the file, service.env.

The following are common tokens that can be used to substitute tokens in the definition of a service object:

MQ_INSTALL_PATH
The location where IBM MQ is installed.

MQ_DATA_PATH
The location of the IBM MQ data directory:

- On UNIX and Linux systems, the IBM MQ data directory location is `/var/mqm/`
- On Windows systems, the location of the IBM MQ data directory is the data directory selected during the installation of IBM MQ

QMNAME
The current queue manager name.

MQ_SERVICE_NAME
The name of the service.

MQ_SERVER_PID
This token can only be used by the STOPARG and STOPCMD arguments.

For server service objects this token is replaced with the process id of the process started by the STARTCMD and STARTARG arguments. Otherwise, this token is replaced with 0.

MQ_Q_MGR_DATA_PATH
The location of the queue manager data directory.

MQ_Q_MGR_DATA_NAME
The transformed name of the queue manager. For more information on name transformation, see Understanding IBM MQ file names.

To use replaceable inserts, insert the token within + characters into any of the STARTCMD, STARTARG, STOPCMD, STOPARG, STDOUT or STDERR strings. For examples of this, see "Examples on using service objects" on page 122.

Examples on using service objects

The services in this section are written with UNIX style path separator characters, except where otherwise stated.

Using a server service object:

This example shows how to define, use, and alter, a server service object to start a trigger monitor.

1. A server service object is defined, using the following MQSC command:

   ```
   DEFINE SERVICE(S1) +
   CONTROL(QMGR) +
   SERVTYPE(SERVER) +
   STARTCMD('+MQ_INSTALL_PATH+bin/runmqtrm') +
   STARTARG('-m +QMNAME+ -q ACCOUNTS.INITIATION.QUEUE') +
   STOPCMD('+MQ_INSTALL_PATH+bin/amqsstop') +
   STOPARG('-m +QMNAME+ -p +MQ_SERVER_PID+')
   ```

   Where:

   `+MQ_INSTALL_PATH+` is a token representing the installation directory.

   `+QMNAME+` is a token representing the name of the queue manager.

   `ACCOUNTS.INITIATION.QUEUE` is the initiation queue.

   `amqsstop` is a sample program provided with IBM MQ which requests the queue manager to break all connections for the process id. `amqsstop` generates PCF commands, therefore the command server must be running.

   `+MQ_SERVER_PID+` is a token representing the process id passed to the stop program.

   See "Replaceable inserts on service definitions" on page 121 for a list of the common tokens.

2. An instance of the server service object will execute when the queue manager is next started. However, we will start an instance of the server service object immediately with the following MQSC command:

   ```
   START SERVICE(S1)
   ```

3. The status of the server service process is displayed, using the following MQSC command:

   ```
   DISPLAY SVSTATUS(S1)
   ```

4. This example now shows how to alter the server service object and have the updates picked up by manually restarting the server service process. The server service object is altered so that the initiation queue is specified as `JUPITER.INITIATION.QUEUE`. The following MQSC command is used:

   ```
   ALTER SERVICE(S1) +
   STARTARG('-m +QMNAME+ -q JUPITER.INITIATION.QUEUE')
   ```

   Note: A running service will not pick up any updates to its service definition until it is restarted.

5. The server service process is restarted so that the alteration is picked up, using the following MQSC commands:

   ```
   STOP SERVICE(S1)
   ```

   Followed by:

   ```
   START SERVICE(S1)
   ```

   The server service process is restarted and picks up the alterations made in 4.

   Note: The MQSC command, `STOP SERVICE`, can only be used if a STOPCMD argument is specified in the service definition.

Using a command service object:

This example shows how to define a command service object to start a program that writes entries to the operating system's system log when a queue manager is started or stopped.

1. The command service object is defined, using the following MQSC command:

   ```
   DEFINE SERVICE(S2) +
   CONTROL(QMGR) +
   SERVTYPE(COMMAND) +
   STARTCMD('/usr/bin/logger') +
   STARTARG('Queue manager +QMNAME+ starting') +
   STOPCMD('/usr/bin/logger') +
   STOPARG('Queue manager +QMNAME+ stopping')
   ```

   Where:

   `logger` is the UNIX and Linux system supplied command to write to the system log.

   `+QMNAME+` is a token representing the name of the queue manager.

Using a command service object when a queue manager ends only:

This example shows how to define a command service object to start a program that writes entries to the operating system's system log when a queue manager is stopped only.

1. The command service object is defined, using the following MQSC command:

   ```
   DEFINE SERVICE(S3) +
   CONTROL(QMGR) +
   SERVTYPE(COMMAND) +
   STOPCMD('/usr/bin/logger') +
   STOPARG('Queue manager +QMNAME+ stopping')
   ```

   Where:

   `logger` is a sample program provided with IBM MQ that can write entries to the operating system's system log.

   `+QMNAME+` is a token representing the name of the queue manager.

More on passing arguments:

This example shows how to define a server service object to start a program called `runserv` when a queue manager is started.

This example is written with Windows style path separator characters.

One of the arguments that is to be passed to the starting program is a string containing a space. This argument needs to be passed as a single string. To achieve this, double quotation marks are used as shown in the following command to define the command service object:

1. The server service object is defined, using the following MQSC command:

   ```
   DEFINE SERVICE(S1) SERVTYPE(SERVER) CONTROL(QMGR) +
   STARTCMD('C:\Program Files\Tools\runserv.exe') +
   STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\"') +
   STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')
   ```

   ```
   DEFINE SERVICE(S4) +
   CONTROL(QMGR) +
   SERVTYPE(SERVER) +
   STARTCMD('C:\Program Files\Tools\runserv.exe') +
   STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\"') +
   STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')
   ```

Where:

+QMNAME+ is a token representing the name of the queue manager.

"C:\Program Files\Tools\" is a string containing a space, which will be passed as a single string.

Autostarting a Service:

This example shows how to define a server service object that can be used to automatically start the Trigger Monitor when the queue manager starts.

1. The server service object is defined, using the following MQSC command:

   ```
   DEFINE SERVICE(TRIG_MON_START) +
   CONTROL(QMGR) +
   SERVTYPE(SERVER) +
   STARTCMD('runmqtrm') +
   STARTARG('-m +QMNAME+ -q +IQNAME+')
   ```

   Where:

   +QMNAME+ is a token representing the name of the queue manager.

   +IQNAME+ is an environment variable defined by the user in one of the service.env files representing the name of the initiation queue.

Managing objects for triggering

IBM MQ enables you to start an application automatically when certain conditions on a queue are met. For example, you might want to start an application when the number of messages on a queue reaches a specified number. This facility is called *triggering*. You have to define the objects that support triggering.

Triggering described in detail in Starting IBM MQ applications using triggers.

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue.

Triggering itself is enabled by the *Trigger* attribute (TRIGGER in MQSC commands). In this example, a trigger event is to be generated when there are 100 messages of priority 5 or greater on the local queue MOTOR.INSURANCE.QUEUE, as follows:

```
DEFINE QLOCAL (MOTOR.INSURANCE.QUEUE) +
PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
MAXMSGL (2000) +
DEFPSIST (YES) +
INITQ (MOTOR.INS.INIT.QUEUE) +
TRIGGER +
TRIGTYPE (DEPTH) +
TRIGDPTH (100)+
TRIGMPRI (5)
```

where:

QLOCAL (MOTOR.INSURANCE.QUEUE)
Is the name of the application queue being defined.

PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
Is the name of the process definition that defines the application to be started by a trigger monitor program.

MAXMSGL (2000)
Is the maximum length of messages on the queue.

DEFPSIST (YES)
Specifies that messages on this queue are persistent by default.

INITQ (MOTOR.INS.INIT.QUEUE)
Is the name of the initiation queue on which the queue manager is to put the trigger message.

TRIGGER
Is the trigger attribute value.

TRIGTYPE (DEPTH)
Specifies that a trigger event is generated when the number of messages of the required priority (TRIGMPRI) reaches the number specified in TRIGDPTH.

TRIGDPTH (100)
Is the number of messages required to generate a trigger event.

TRIGMPRI (5)
Is the priority of messages that are to be counted by the queue manager in deciding whether to generate a trigger event. Only messages with priority 5 or higher are counted.

Defining an initiation queue

When a trigger event occurs, the queue manager puts a trigger message on the initiation queue specified in the application queue definition. Initiation queues have no special settings, but you can use the following definition of the local queue MOTOR.INS.INIT.QUEUE for guidance:

```
DEFINE QLOCAL(MOTOR.INS.INIT.QUEUE) +
GET (ENABLED) +
NOSHARE +
NOTRIGGER +
MAXMSGL (2000) +
MAXDEPTH (1000)
```

Defining a process

Use the DEFINE PROCESS command to create a process definition. A process definition defines the application to be used to process messages from the application queue. The application queue definition names the process to be used and thereby associates the application queue with the application to be used to process its messages. This is done through the PROCESS attribute on the application queue MOTOR.INSURANCE.QUEUE. The following MQSC command defines the required process, MOTOR.INSURANCE.QUOTE.PROCESS, identified in this example:

```
DEFINE PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
DESCR ('Insurance request message processing') +
APPLTYPE (UNIX) +
APPLICID ('/u/admin/test/IRMP01') +
USERDATA ('open, close, 235')
```

Where:

MOTOR.INSURANCE.QUOTE.PROCESS
Is the name of the process definition.

DESCR ('Insurance request message processing')
Describes the application program to which this definition relates. This text is displayed when you use the DISPLAY PROCESS command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotation marks.

APPLTYPE (UNIX)
Is the type of application to be started.

APPLICID ('/u/admin/test/IRMP01')
Is the name of the application executable file, specified as a fully qualified file name. In Windows systems, a typical `APPLICID` value would be `c:\appl\test\irmp01.exe`.

USERDATA ('open, close, 235')
Is user-defined data, which can be used by the application.

Displaying attributes of a process definition

Use the DISPLAY PROCESS command to examine the results of your definition. For example:

```
DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)


24 : DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL
AMQ8407: Display Process details.
DESCR ('Insurance request message processing')
APPLICID ('/u/admin/test/IRMP01')
USERDATA (open, close, 235)
PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
APPLTYPE (UNIX)
```

You can also use the MQSC command ALTER PROCESS to alter an existing process definition, and the DELETE PROCESS command to delete a process definition.

Using the `dmpmqmsg` utility between two systems

The **dmpmqmsg** utility (formerly *qload*) is incorporated into the product in Version 8.0. Formerly the utility has been available as SupportPac MO03.

Overview

The **dmpmqmsg** utility allows you to copy or move the contents of a queue, or its messages, to a file. This file can be saved away as required and used at some later point to reload the messages back onto the queue.

Important: The file has a specific format understood by the utility. However, the file is human-readable, so that you can update it in an editor before you reload it. If you do edit the file you must not change its format.

Possible uses are:
- Saving the messages that are on a queue, to a file. Possibly for archiving purposes, and later reload back to a queue.
- Reloading a queue with messages you previously saved to a file.
- Removing old messages from a queue.
- 'Replaying' test messages from a stored location, even maintaining the correct time between the messages if required.

Attention: SupportPac MO03 used the **-l** parameter for specifying local or client binding. **-l** has been replaced by the **-c** parameter.

-P is now used for codepage information instead of **-c**.

See dmpmqmsg for further information on the command and the available parameters.

Example of using the `dmpmqmsg` utility on Linux, using a Windows machine

You have a queue manager on a Linux machine that has messages on a queue (*Q1*) that you want to move into another queue (*Q2*) in the same queue manager. You want to initiate the **dmpmqmsg** utility from a Windows machine.

Queue (*Q1*) has four messages that have been added by using the sample **amqsput** (local queue manager) or **amqsputc** (remote queue manager) application.

On the Linux machine you see:

```
display ql(Q1) CURDEPTH
        2 : display ql(Q1) CURDEPTH
AMQ8409: Display Queue details.
     QUEUE(Q1)
TYPE(QLOCAL)
     CURDEPTH(4)
```

Set the MQSERVER environment variable to point to the queue manager in Linux. For example:

```
set MQSERVER=SYSTEM.DEF.SVRCONN/TCP/veracruz.x.com(1414)
```

where *veracruz* is the name of the machine.

Run the **dmpmqmsg** utility to read from the queue, *Q1*, and store the output in `c:\temp\mqqload.txt`.

Connect as a remote client to the queue manager, *QM_VER*, running in the Linux host and port established by MQSERVER. You achieve the connection as a remote client by using the attribute: `-c`.

```
dmpmqmsg -m QM_VER -i Q1 -f c:\temp\mqqload.txt -c
Read      - Files:    0   Messages:      4   Bytes:           22
Written - Files:    1   Messages:      4   Bytes:           22
```

The output file `c:\temp\mqqload.txt` contains text, using a format that the **dmpmqmsg** utility understands.

On the Windows machine, issue the **dmpmqmsg** command (using the `-o` option instead of the `-i` option) to load queue (*Q2*) on the Linux machine from a file on the Windows machine:

```
dmpmqmsg -m QM_VER -o Q2 -f c:\temp\mqqload.txt -c
Read      - Files:    1   Messages:      4   Bytes:           22
Written - Files:    0   Messages:      4   Bytes:           22
```

On the Linux machine, note that there are now four messages in the queue that have been restored from the file.

```
display ql(Q2) CURDEPTH
        6 : display ql(Q2) CURDEPTH
AMQ8409: Display Queue details.
     QUEUE(Q2)
TYPE(QLOCAL)
     CURDEPTH(4)
```

On the Linux machine,

Delete the messages from the original queue.

```
clear qlocal(Q1)
        4 : clear qlocal(Q1)
AMQ8022: IBM MQ queue cleared.
```

Confirm that there are no more messages on the original queue:

```
display ql(Q1) CURDEPTH
        5 : display ql(Q1) CURDEPTH
AMQ8409: Display Queue details.
     QUEUE(Q1)
TYPE(QLOCAL)
     CURDEPTH(0)
```

See dmpmqmsg for a description of the command and its parameters.

Related concepts:

"Examples of using the **dmpmqmsg** utility"
Simple ways in which you can use the **dmpmqmsg** utility (formerly *qload*). This utility is incorporated into the product in Version 8.0.

Examples of using the `dmpmqmsg` utility

Simple ways in which you can use the **dmpmqmsg** utility (formerly *qload*). This utility is incorporated into the product in Version 8.0.

Formerly the utility has been available as SupportPac MO03.

Unload a queue to a file

Use the following options on the command line to save the messages that are on a queue, into a file:

```
dmpmqmsg -m QM1 -i Q1 -f c:\myfile
```

This command takes a copy of the messages from the queue and saves them in the file specified.

Unload a queue to a series of files

You can unload a queue to a series of files by using an `insert` character in the file name. In this mode each message is written to a new file:

```
dmpmqmsg -m QM1 -i Q1 -f c:\myfile%n
```

This command unloads the queue to files, `myfile1`, `myfile2`, `myfile3`, and so on.

Load a queue from a file

To reload a queue with the messages you saved in "Unload a queue to a file," use the following options on the command line:

```
dmpmqmsg -m QM1 -o Q1 -f c:\myfile%n
```

This command unloads the queue to files, `myfile1`, `myfile2`, `myfile3`, and so on.

Load a queue from a series of files

You can load a queue from a series of files by using an `insert` character in the file name. In this mode each message is written to a new file:

```
dmpmqmsg -m QM1 -o Q1 -f c:\myfile%n
```

This command loads the queue to files, `myfile1`, `myfile2`, `myfile3`, and so on.

Copy the messages from one queue to another queue

Replace the file parameter in "Unload a queue to a file," with another queue name and use the following options:

```
dmpmqmsg -m QM1 -i Q1 -o Q2
```

This command allows the messages from one queue to be copied to another queue.

Copy the first 100 messages from one queue to another queue

Use the command in the previous example and add the `-r#10` option:

```
dmpmqmsg -m QM1 -i Q1 -o Q2 -r#10
```

Move the messages from one queue to another queue

A variation on "Load a queue from a file" on page 128. Note the distinction between using **-i** (lower case) which only browses a queue, and **-I** (upper case) which destructively gets from a queue:

```
dmpmqmsg -m QM1 -I Q1 -o Q2
```

Move messages older than one day from one queue to another queue

This example shows the use of age selection. Messages can be selected that are older than, younger than, or within a range of ages.

```
dmpmqmsg -m QM1 -I Q1 -o Q2 -T1440
```

Display the ages of messages currently on a queue

Use the following options on the command line:

```
dmpmqmsg -m QM1 -i Q1 -f stdout -dT
```

Work with the message file

Having unloaded the message from your queue, as in "Unload a queue to a file" on page 128, you might want to edit the file.

You might also want to change the format of the file to use one of the display options that you did not specify at the time you unloaded the queue.

You can use the **dmpmqmsg** utility to reprocess the file into the desired format even after the unload of the queue has taken place. Use the following options on the command line.

```
dmpmqmsg -f c:\oldfile -f c:\newfile -dA
```

See dmpmqmsg for a description of the command and its parameters.

Administering remote IBM MQ objects

This section tells you how to administer IBM MQ objects on a remote queue manager using MQSC commands, and how to use remote queue objects to control the destination of messages and reply messages.

This section describes:
- "Channels, clusters, and remote queuing" on page 130
- "Remote administration from a local queue manager" on page 131
- "Creating a local definition of a remote queue" on page 137
- "Using remote queue definitions as aliases" on page 142
- "Data conversion" on page 143

Channels, clusters, and remote queuing

A queue manager communicates with another queue manager by sending a message and, if required, receiving back a response. The receiving queue manager could be:

- On the same machine
- On another machine in the same location (or even on the other side of the world)
- Running on the same platform as the local queue manager
- Running on another platform supported by IBM MQ

These messages might originate from:

- User-written application programs that transfer data from one node to another
- User-written administration applications that use PCF commands or the MQAI
- The IBM MQ Explorer.
- Queue managers sending:
  - Instrumentation event messages to another queue manager
  - MQSC commands issued from a **runmqsc** command in indirect mode (where the commands are run on another queue manager)

Before a message can be sent to a remote queue manager, the local queue manager needs a mechanism to detect the arrival of messages and transport them consisting of:

- At least one channel
- A transmission queue
- A channel initiator

For a remote queue manager to received a message, a listener is required.

A channel is a one-way communication link between two queue managers and can carry messages destined for any number of queues at the remote queue manager.

Each end of the channel has a separate definition. For example, if one end is a sender or a server, the other end must be a receiver or a requester. A simple channel consists of a *sender channel definition* at the local queue manager end and a *receiver channel definition* at the remote queue manager end. The two definitions must have the same name and together constitute a single message channel.

If you want the remote queue manager to respond to messages sent by the local queue manager, set up a second channel to send responses back to the local queue manager.

Use the MQSC command DEFINE CHANNEL to define channels. In this section, the examples relating to channels use the default channel attributes unless otherwise specified.

There is a message channel agent (MCA) at each end of a channel, controlling the sending and receiving of messages. The MCA takes messages from the transmission queue and puts them on the communication link between the queue managers.

A transmission queue is a specialized local queue that temporarily holds messages before the MCA picks them up and sends them to the remote queue manager. You specify the name of the transmission queue on a *remote queue definition*.

You can allow an MCA to transfer messages using multiple threads. This process is known as *pipelining*. Pipelining enables the MCA to transfer messages more efficiently, improving channel performance. See Attributes of channels for details of how to configure a channel to use pipelining.

"Preparing channels and transmission queues for remote administration" on page 133 tells you how to use these definitions to set up remote administration.

For more information about setting up distributed queuing in general, see Distributed queuing components.

Remote administration using clusters

In an IBM MQ network using distributed queuing, every queue manager is independent. If one queue manager needs to send messages to another queue manager, it must define a transmission queue, a channel to the remote queue manager, and a remote queue definition for every queue to which it wants to send messages.

A *cluster* is a group of queue managers set up in such a way that the queue managers can communicate directly with one another over a single network without complex transmission queue, channel, and queue definitions. Clusters can be set up easily, and typically contain queue managers that are logically related in some way and need to share data or applications. Even the smallest cluster reduces system administration costs.

Establishing a network of queue managers in a cluster involves fewer definitions than establishing a traditional distributed queuing environment. With fewer definitions to make, you can set up or change your network more quickly and easily, and reduce the risk of making an error in your definitions.

To set up a cluster, you need one cluster sender (CLUSSDR) and one cluster receiver (CLUSRCVR) definition for each queue manager. You do not need any transmission queue definitions or remote queue definitions. The principles of remote administration are the same when used within a cluster, but the definitions themselves are greatly simplified.

Remote administration from a local queue manager

This section tells you how to administer a remote queue manager from a local queue manager using MQSC and PCF commands.

Preparing the queues and channels is essentially the same for both MQSC and PCF commands. In this section, the examples show MQSC commands, because they are easier to understand. For more information about writing administration programs using PCF commands, see "Using Programmable Command Formats" on page 7.

You send MQSC commands to a remote queue manager either interactively or from a text file containing the commands. The remote queue manager might be on the same machine or, more typically, on a different machine. You can remotely administer queue managers in other IBM MQ environments, including UNIX and Linux systems, Windows systems, IBM i, and z/OS.

To implement remote administration, you must create specific objects. Unless you have specialized requirements, the default values (for example, for maximum message length) are sufficient.

Preparing queue managers for remote administration

How to use MQSC commands to prepare queue managers for remote administration.

Figure 16 shows the configuration of queue managers and channels that you need for remote administration using the **runmqsc** command. The object `source.queue.manager` is the source queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned. The object `target.queue.manager` is the name of the target queue manager, which processes the commands and generates any operator messages.

Note: If you are using **runmqsc** with the -w option, `source.queue.manager` ***must*** be the default queue manager. For further information on creating a queue manager, see crtmqm.

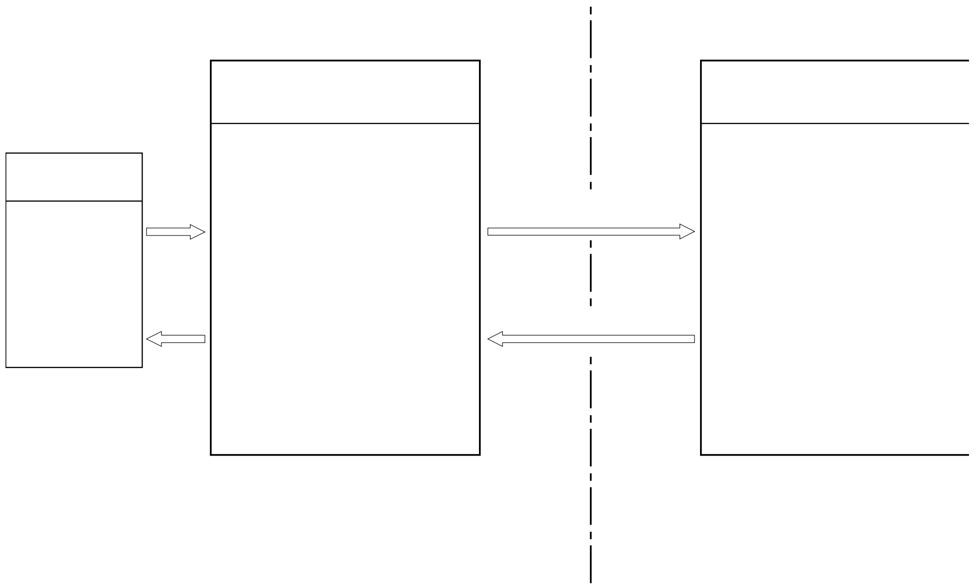


Figure 16. Remote administration using MQSC commands

On both systems, if you have not already done so:

- Create the queue manager and the default objects, using the **crtmqm** command.
- Start the queue manager, using the **strmqm** command.

On the target queue manager:

- The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, must be present. This queue is created by default when a queue manager is created.

You have to run these commands locally or over a network facility such as Telnet.

Preparing channels and transmission queues for remote administration

How to use MQSC commands to prepare channels and transmission queues for remote administration.

To run MQSC commands remotely, set up two channels, one for each direction, and their associated transmission queues. This example assumes that you are using TCP/IP as the transport type and that you know the TCP/IP address involved.

The channel `source.to.target` is for sending MQSC commands from the source queue manager to the target queue manager. Its sender is at `source.queue.manager` and its receiver is at `target.queue.manager`. The channel `target.to.source` is for returning the output from commands and any operator messages that are generated to the source queue manager. You must also define a transmission queue for each channel. This queue is a local queue that is given the name of the receiving queue manager. The XMITQ name must match the remote queue manager name in order for remote administration to work, unless you are using a queue manager alias. Figure 17 summarizes this configuration.

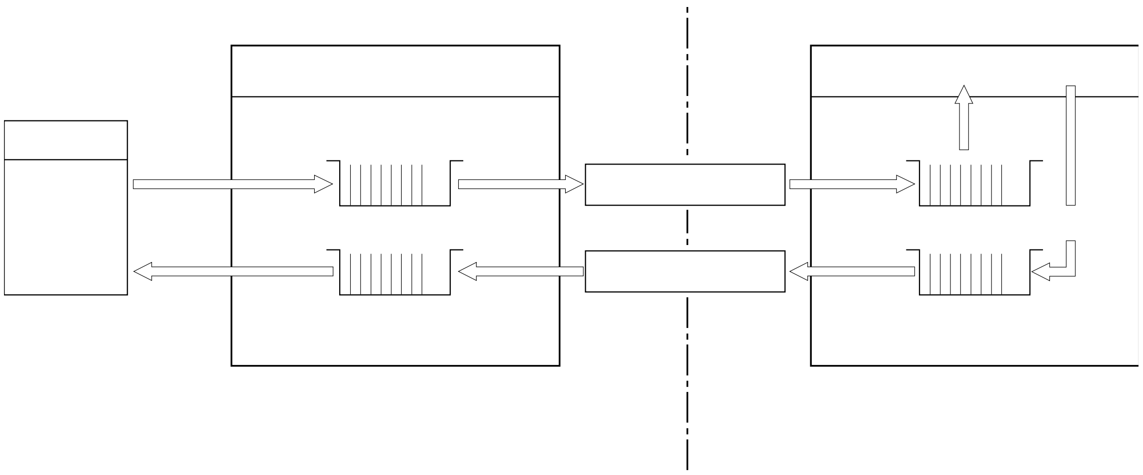


Figure 17. Setting up channels and queues for remote administration

See Configuring distributed queuing for more information about setting up channels.

Defining channels, listeners, and transmission queues:

On the source queue manager (`source.queue.manager`), issue the following MQSC commands to define the channels, listener, and the transmission queue:

1. Define the sender channel at the source queue manager:
```
DEFINE CHANNEL ('source.to.target') +
CHLTYPE(SDR) +
CONNAME (RHX5498) +
XMITQ ('target.queue.manager') +
TRPTYPE(TCP)
```
2. Define the receiver channel at the source queue manager:
```
DEFINE CHANNEL ('target.to.source') +
CHLTYPE(RCVR) +
TRPTYPE(TCP)
```
3. Define the listener on the source queue manager:
```
DEFINE LISTENER ('source.queue.manager') +
TRPTYPE (TCP)
```
4. Define the transmission queue on the source queue manager:
```
DEFINE QLOCAL ('target.queue.manager') +
USAGE (XMITQ)
```

Issue the following commands on the target queue manager (`target.queue.manager`), to create the channels, listener, and the transmission queue:

1. Define the sender channel on the target queue manager:
   ```
   DEFINE CHANNEL ('target.to.source') +
   CHLTYPE(SDR) +
   CONNAME (RHX7721) +
   XMITQ ('source.queue.manager') +
   TRPTYPE(TCP)
   ```
2. Define the receiver channel on the target queue manager:
   ```
   DEFINE CHANNEL ('source.to.target') +
   CHLTYPE(RCVR) +
   TRPTYPE(TCP)
   ```
3. Define the listener on the target queue manager:
   ```
   DEFINE LISTENER ('target.queue.manager') +
   TRPTYPE (TCP)
   ```
4. Define the transmission queue on the target queue manager:
   ```
   DEFINE QLOCAL ('source.queue.manager') +
   USAGE (XMITQ)
   ```

Note: The TCP/IP connection names specified for the CONNAME attribute in the sender channel definitions are for illustration only. This is the network name of the machine at the *other* end of the connection. Use the values appropriate for your network.

Starting the listeners and channels:

How to use MQSC commands to start listeners and channels.

Start both listeners by using the following MQSC commands:

1. Start the listener on the source queue manager, `source.queue.manager`, by issuing the following MQSC command:
   ```
   START LISTENER ('source.queue.manager')
   ```
2. Start the listener on the target queue manager, `target.queue.manager`, by issuing the following MQSC command:
   ```
   START LISTENER ('target.queue.manager')
   ```

Start both sender channels by using the following MQSC commands:

1. Start the sender channel on the source queue manager, `source.queue.manager`, by issuing the following MQSC command:
   ```
   START CHANNEL ('source.to.target')
   ```
2. Start the sender channel on the target queue manager, `target.queue.manager`, by issuing the following MQSC command:
   ```
   START CHANNEL ('target.to.source')
   ```

Automatic definition of channels:

You enable automatic definition of receiver and server-connection definitions by updating the queue manager object using the MQSC command, ALTER QMGR (or the PCF command Change Queue Manager).

If IBM MQ receives an inbound attach request and cannot find an appropriate receiver or server-connection channel, it creates a channel automatically. Automatic definitions are based on two default definitions supplied with IBM MQ: SYSTEM.AUTO.RECEIVER and SYSTEM.AUTO.SVRCONN.

For more information about creating channel definitions automatically, see Preparing channels. For information about automatically defining channels for clusters, see Working with auto-defined channels.

Managing the command server for remote administration

How to start, stop, and display the status of the command server. A command server is mandatory for all administration involving PCF commands, the MQAI, and also for remote administration.

Each queue manager can have a command server associated with it. A command server processes any incoming commands from remote queue managers, or PCF commands from applications. It presents the commands to the queue manager for processing and returns a completion code or operator message depending on the origin of the command.

Note: For remote administration, ensure that the target queue manager is running. Otherwise, the messages containing commands cannot leave the queue manager from which they are issued. Instead, these messages are queued in the local transmission queue that serves the remote queue manager. Avoid this situation.

There are separate control commands for starting and stopping the command server. Providing the command server is running, users of IBM MQ for Windows or IBM MQ for Linux (x86 and x86-64 platforms) can perform the operations described in the following sections using the IBM MQ Explorer. For more information, see “Administration using the MQ Explorer” on page 56.

Starting the command server

Depending on the value of the queue manager attribute, *SCMDSERV*, the command server is either started automatically when the queue manager starts, or must be started manually. The value of the queue manager attribute can be altered using the MQSC command `ALTER QMGR` specifying the parameter `SCMDSERV`. By default, the command server is started automatically.

If *SCMDSERV* is set to MANUAL, start the command server using the command:

```
strmqcsv saturn.queue.manager
```

where `saturn.queue.manager` is the queue manager for which the command server is being started.

Displaying the status of the command server

For remote administration, ensure that the command server on the target queue manager is running. If it is not running, remote commands cannot be processed. Any messages containing commands are queued in the target queue manager's command queue.

To display the status of the command server for a queue manager, issue the following MQSC command:

```
DISPLAY QMSTATUS CMDSERV
```

Stopping a command server

To end the command server started by the previous example use the following command:

```
endmqcsv saturn.queue.manager
```

You can stop the command server in two ways:
- For a controlled stop, use the **endmqcsv** command with the -c flag, which is the default.
- For an immediate stop, use the **endmqcsv** command with the -i flag.

Note: Stopping a queue manager also ends the command server associated with it.

Issuing MQSC commands on a remote queue manager

You can use a particular form of the `runmqsc` command to run MQSC commands on a remote queue manager.

The command server ***must*** be running on the target queue manager, if it is going to process MQSC commands remotely. (This is not necessary on the source queue manager). For information on how to start the command server on a queue manager, see "Managing the command server for remote administration" on page 135.

On the source queue manager, you can then run MQSC commands interactively in indirect mode by typing:

```
runmqsc -w 30 target.queue.manager
```

This form of the **runmqsc** command, with the -w flag, runs the MQSC commands in indirect mode, where commands are put (in a modified form) on the command server input queue and executed in order.

When you type in an MQSC command, it is redirected to the remote queue manager, in this case, `target.queue.manager`. The timeout is set to 30 seconds; if a reply is not received within 30 seconds, the following message is generated on the local (source) queue manager:

```
AMQ8416: MQSC timed out waiting for a response from the command server.
```

When you stop issuing MQSC commands, the local queue manager displays any timed-out responses that have arrived and discards any further responses.

The source queue manager defaults to the default local queue manager. If you specify the `-m` *LocalQmgrName* option in the `runmqsc` command, you can direct the commands to be issued by way of any local queue manager.

In indirect mode, you can also run an MQSC command file on a remote queue manager. For example:

```
runmqsc -w 60 target.queue.manager < mycomds.in > report.out
```

where `mycomds.in` is a file containing MQSC commands and `report.out` is the report file.

Suggested method for issuing commands remotely

When you are issuing commands on a remote queue manager, consider using the following approach:
1. Put the MQSC commands to be run on the remote system in a command file.
2. Verify your MQSC commands locally, by specifying the -v flag on the **runmqsc** command.
   You cannot use **runmqsc** to verify MQSC commands on another queue manager.
3. Check that the command file runs locally without error.
4. Run the command file on the remote system.

If you have problems using MQSC commands remotely

If you have difficulty in running MQSC commands remotely, make sure that you have:
- Started the command server on the target queue manager.
- Defined a valid transmission queue.
- Defined the two ends of the message channels for both:
  - The channel along which the commands are being sent.
  - The channel along which the replies are to be returned.
- Specified the correct connection name (CONNAME) in the channel definition.
- Started the listeners before you started the message channels.
- Checked that the disconnect interval has not expired, for example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.
- Sent requests from a source queue manager that do not make sense to the target queue manager (for example, requests that include parameters that are not supported on the remote queue manager).

See also "Resolving problems with MQSC commands" on page 81.

Working with queue managers on z/OS:

You can issue MQSC commands to a z/OS queue manager from a queue manager on the platforms described in this guide. However, to do this, you must modify the **runmqsc** command and the channel definitions at the sender.

In particular, you add the -x flag to the **runmqsc** command on the source node to specify that the target queue manager is running under z/OS:

```
runmqsc -w 30 -x target.queue.manager
```

Creating a local definition of a remote queue

A local definition of a remote queue is a definition on a local queue manager that refers to a queue on a remote queue manager.

You do not have to define a remote queue from a local position, but the advantage of doing so is that applications can refer to the remote queue by its locally-defined name instead of having to specify a name that is qualified by the ID of the queue manager on which the remote queue is located.

Understanding how local definitions of remote queues work

An application connects to a local queue manager and then issues an **MQOPEN** call. In the open call, the queue name specified is that of a remote queue definition on the local queue manager. The remote queue definition supplies the names of the target queue, the target queue manager, and optionally, a transmission queue. To put a message on the remote queue, the application issues an **MQPUT** call, specifying the handle returned from the **MQOPEN** call. The queue manager uses the remote queue name and the remote queue manager name in a transmission header at the start of the message. This information is used to route the message to its correct destination in the network.

As administrator, you can control the destination of the message by altering the remote queue definition.

The following example shows how an application puts a message on a queue owned by a remote queue manager. The application connects to a queue manager, for example, `saturn.queue.manager`. The target queue is owned by another queue manager.

On the **MQOPEN** call, the application specifies these fields:

Field value	Description
ObjectName CYAN.REMOTE.QUEUE	Specifies the local name of the remote queue object. This defines the target queue and the target queue manager.
ObjectType (Queue)	Identifies this object as a queue.
ObjectQmgrName Blank or `saturn.queue.manager`	This field is optional. If blank, the name of the local queue manager is assumed. (This is the queue manager on which the remote queue definition exists.)

After this, the application issues an **MQPUT** call to put a message onto this queue.

On the local queue manager, you can create a local definition of a remote queue using the following MQSC commands:

```
DEFINE QREMOTE (CYAN.REMOTE.QUEUE) +
DESCR ('Queue for auto insurance requests from the branches') +
RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE) +
RQMNAME (jupiter.queue.manager) +
XMITQ (INQUOTE.XMIT.QUEUE)
```

where:

`QREMOTE (CYAN.REMOTE.QUEUE)`
Specifies the local name of the remote queue object. This is the name that applications connected to this queue manager must specify in the **MQOPEN** call to open the queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE on the remote queue manager `jupiter.queue.manager`.

`DESCR ('Queue for auto insurance requests from the branches')`
Provides additional text that describes the use of the queue.

`RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)`
Specifies the name of the target queue on the remote queue manager. This is the real target queue for messages sent by applications that specify the queue name CYAN.REMOTE.QUEUE. The queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE must be defined as a local queue on the remote queue manager.

`RQMNAME (jupiter.queue.manager)`
Specifies the name of the remote queue manager that owns the target queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE.

`XMITQ (INQUOTE.XMIT.QUEUE)`
Specifies the name of the transmission queue. This is optional; if the name of a transmission queue is not specified, a queue with the same name as the remote queue manager is used.

In either case, the appropriate transmission queue must be defined as a local queue with a *Usage* attribute specifying that it is a transmission queue (USAGE(XMITQ) in MQSC commands).

An alternative way of putting messages on a remote queue

Using a local definition of a remote queue is not the only way of putting messages on a remote queue. Applications can specify the full queue name, including the remote queue manager name, as part of the **MQOPEN** call. In this case, you do not need a local definition of a remote queue. However, this means that applications must either know, or have access to, the name of the remote queue manager at run time.

Using other commands with remote queues

You can use MQSC commands to display or alter the attributes of a remote queue object, or you can delete the remote queue object. For example:

- To display the remote queue's attributes:
  ```
  DISPLAY QUEUE (CYAN.REMOTE.QUEUE)
  ```
- To change the remote queue to enable puts. This does not affect the target queue, only applications that specify this remote queue:
  ```
  ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)
  ```
- To delete this remote queue. This does not affect the target queue, only its local definition:
  ```
  DELETE QREMOTE (CYAN.REMOTE.QUEUE)
  ```

Note: When you delete a remote queue, you delete only the local representation of the remote queue. You do not delete the remote queue itself or any messages on it.

Defining a transmission queue

A transmission queue is a local queue that is used when a queue manager forwards messages to a remote queue manager through a message channel.

The channel provides a one-way link to the remote queue manager. Messages are queued at the transmission queue until the channel can accept them. When you define a channel, you must specify a transmission queue name at the sending end of the message channel.

The MQSC command attribute USAGE defines whether a queue is a transmission queue or a normal queue.

Default transmission queues

When a queue manager sends messages to a remote queue manager, it identifies the transmission queue using the following sequence:

1. The transmission queue named on the XMITQ attribute of the local definition of a remote queue.
2. A transmission queue with the same name as the target queue manager. (This value is the default value on XMITQ of the local definition of a remote queue.)
3. The transmission queue named on the DEFXMITQ attribute of the local queue manager.

For example, the following MQSC command creates a default transmission queue on `source.queue.manager` for messages going to `target.queue.manager`:

```
DEFINE QLOCAL ('target.queue.manager') +
DESCR ('Default transmission queue for target qm') +
USAGE (XMITQ)
```

Applications can put messages directly on a transmission queue, or indirectly through a remote queue definition. See also “Creating a local definition of a remote queue” on page 137.

Checking that async commands for distributed networks have finished

Many commands are asynchronous when used in a distributed network. Depending on the command, and the network state when it is issued, it can take a significant amount of time to finish. The queue manager does not issue a message on completion, so you need other ways of checking that the command has finished.

About this task

Almost any configuration change that you make to a cluster is likely to complete asynchronously. This is because of the internal administration and updating cycles that operate within clusters. For publish/subscribe hierarchies, any configuration change that affects subscriptions is likely to complete asynchronously. This is not always obvious from the name of the command.

The following MQSC commands might all complete asynchronously. Each of these commands has a PCF equivalent, and most are also available from within MQ Explorer . When run on a small network with no workload, these commands typically complete within a few seconds. However, this is not the case for larger and busier networks. Also, the **`REFRESH CLUSTER`** command might take much longer, particularly when it is issued on multiple queue managers at the same time.

To have confidence that these commands have finished, check that the expected objects exist on the remote queue managers.

Procedure

- ALTER QMGR

  For the ALTER QMGR PARENT command, use `DISPLAY PUBSUB TYPE(PARENT) ALL` to track the status of the requested parent relationship.

  For the ALTER QMGR REPOS and ALTER QMGR REPOSNL commands, use `DISPLAY CLUSQMGR QMTYPE` to confirm completion.
- DEFINE CHANNEL, ALTER CHANNEL, and DELETE CHANNEL

  For all parameters listed in the table ALTER CHANNEL parameters, use the `DISPLAY CLUSQMGR` command to monitor when changes have been propagated to the cluster.
- DEFINE NAMELIST, ALTER NAMELIST, and DELETE NAMELIST.

  If you use a **`NAMELIST`** on the **`CLUSNL`** attribute of a **`QMgr`** object, a queue or a cluster channel might affect that object. Monitor as appropriate for the affected object.

  Changes to `SYSTEM.QPUBSUB.QUEUE.NAMELIST` might affect creation or cancellation of proxy subscriptions in a publish/subscribe hierarchy. Use the `DISPLAY SUB SUBTYPE(PROXY)` command to monitor this.
- DEFINE queues, ALTER queues, and DELETE queues.

  For all parameters listed in the table Parameters that can be returned by the DISPLAY QUEUE command, use the `DISPLAY QCLUSTER` command to monitor when changes have been propagated to the cluster.
- DEFINE SUB, and DELETE SUB

  When you define the first subscription on a topic string, you might create proxy subscriptions in a publish/subscribe hierarchy or publish/subscribe cluster. Similarly, when you delete the last subscription on a topic string, you might cancel proxy subscriptions in a publish/subscribe hierarchy or publish/subscribe cluster.

  To check that a command defining or deleting a subscription has finished, check whether or not the expected proxy subscription exists on other queue managers in the distributed network. If you are using *direct routing* in a cluster, check that the expected proxy subscription exists on the other partial repositories in the cluster. If you are using *topic host routing* in a cluster, check that the expected proxy subscription exists on the matching topic hosts. Use the following MQSC command:

  ```
  DISPLAY SUB(*) SUBTYPE(PROXY)
  ```

Use the same check for the following equivalent subscribe and unsubscribe MQI calls, when they are issued in a cluster or hierarchy:

– Subscribe by using MQSUB.
– Unsubscribe by using MQCLOSE with MQCO_REMOVE_SUB.

- DEFINE TOPIC, ALTER TOPIC, and DELETE TOPIC

  To check that a command defining, altering or deleting a clustered topic has finished, display the topic in the other partial repositories in the cluster (if you are using *direct routing*) or on the other topic hosts (if you are using *topic host routing*).

  For all parameters listed in the table Parameters that can be returned by the DISPLAY TOPIC command, use the `DISPLAY TCLUSTER` command to monitor when changes have been propagated to the cluster.

  Note:

  – The **CLUSTER** parameter can affect creation or cancellation of proxy subscriptions in a publish/subscribe cluster.
  – The **PROXYSUB** and **SUBSCOPE** parameters can affect creation or cancellation of proxy subscriptions in a publish/subscribe hierarchy or publish/subscribe cluster.
  – Use the `DISPLAY SUB SUBTYPE(PROXYSUB)` command to monitor this.

- REFRESH CLUSTER

  If you are running the **REFRESH CLUSTER** command, poll the cluster command queue depth. Wait for it to reach zero, and remain at zero, before looking for the objects.

  1. Use the following MQSC command to check that the cluster command queue depth is zero.
     ```
     DISPLAY QL(SYSTEM.CLUSTER.COMMAND.QUEUE) CURDEPTH
     ```
  2. Repeat the check until the queue depth reaches zero, and remains at zero in the subsequent check.

  The **REFRESH CLUSTER** command removes and recreates objects, and in large configurations can take a significant time to complete. See REFRESH CLUSTER considerations for publish/subscribe clusters.

- REFRESH QMGR TYPE(PROXYSUB)

  To check that the **REFRESH QMGR TYPE(PROXYSUB)** command has finished, check that the proxy subscriptions have been corrected on other queue managers in the distributed network. If you are using *direct routing* in a cluster, check that the proxy subscriptions have been corrected on the other partial repositories in the cluster. If you are using *topic host routing* in a cluster, check that the expected proxy subscriptions have been corrected on the matching topic hosts. Use the following MQSC command:

  ```
  DISPLAY SUB(*) SUBTYPE(PROXYSUB)
  ```

- RESET CLUSTER

  To check that the **RESET CLUSTER** command has completed, use `DISPLAY CLUSQMGR`.

- RESET QMGR TYPE(PUBSUB)

  To check that the **RESET QMGR** command has completed, use `DISPLAY PUBSUB TYPE(PARENT|CHILD)`.

  Note: The **RESET QMGR** command might cause cancellation of proxy subscriptions in a publish/subscribe hierarchy or publish/subscribe cluster. Use the `DISPLAY SUB SUBTYPE(PROXYSUB)` command to monitor this.

- You might also want to monitor other system queues that, as and when commands complete, tend towards a queue depth of zero. For example, you might want to monitor the `SYSTEM.INTER.QMGR.CONTROL` queue, and the `SYSTEM.INTER.QMGR.FANREQ` queue. See Monitoring proxy subscription traffic in clusters and Balancing producers and consumers in publish/subscribe networks.

What to do next

If these checks do not confirm that an asynchronous command has finished, an error might have occurred. To investigate, first check the log for the queue manager on which the command was issued, then (for a cluster) check the cluster full repository logs.

Related information:

z/OS Asynchronous behavior of CLUSTER commands on z/OS

Using remote queue definitions as aliases

In addition to locating a queue on another queue manager, you can also use a local definition of a remote queue for Queue manager aliases and reply-to queue aliases. Both types of alias are resolved through the local definition of a remote queue. You must set up the appropriate channels for the message to arrive at its destination.

Queue manager aliases

An alias is the process by which the name of the target queue manager, as specified in a message, is modified by a queue manager on the message route. Queue manager aliases are important because you can use them to control the destination of messages within a network of queue managers.

You do this by altering the remote queue definition on the queue manager at the point of control. The sending application is not aware that the queue manager name specified is an alias.

For more information about queue manager aliases, see What are aliases?.

Reply-to queue aliases

Optionally, an application can specify the name of a reply-to queue when it puts a *request message* on a queue.

If the application that processes the message extracts the name of the reply-to queue, it knows where to send the *reply message*, if required.

A reply-to queue alias is the process by which a reply-to queue, as specified in a request message, is altered by a queue manager on the message route. The sending application is not aware that the reply-to queue name specified is an alias.

A reply-to queue alias lets you alter the name of the reply-to queue and optionally its queue manager. This in turn lets you control which route is used for reply messages.

For more information about request messages, reply messages, and reply-to queues, see Types of message and Reply-to queue and queue manager.

For more information about reply-to queue aliases, see Reply-to queue aliases and clusters.

Data conversion

Message data in IBM MQ defined formats (also known as *built-in formats*) can be converted by the queue manager from one coded character set to another, provided that both character sets relate to a single language or a group of similar languages.

For example, conversion between coded character sets with identifiers (CCSIDs) 850 and 500 is supported, because both apply to Western European languages.

For EBCDIC newline (NL) character conversions to ASCII, see All queue managers.

Supported conversions are defined in Data conversion.

When a queue manager cannot convert messages in built-in formats

The queue manager cannot automatically convert messages in built-in formats if their CCSIDs represent different national-language groups. For example, conversion between CCSID 850 and CCSID 1025 (which is an EBCDIC coded character set for languages using Cyrillic script) is not supported because many of the characters in one coded character set cannot be represented in the other. If you have a network of queue managers working in different national languages, and data conversion among some of the coded character sets is not supported, you can enable a default conversion. Default data conversion is described in "Default data conversion."

File ccsid.tbl

The file ccsid.tbl is used for the following purposes:

- In IBM MQ for Windows it records all the supported code sets.
- On AIX and HP-UX platforms, the supported code sets are held internally by the operating system.
- For all other UNIX and Linux platforms, the supported code sets are held in conversion tables provided by IBM MQ.
- It specifies any additional code sets. To specify additional code sets, you need to edit ccsid.tbl (guidance on how to do this is provided in the file).
- It specifies any default data conversion.

You can update the information recorded in ccsid.tbl; you might want to do this if, for example, a future release of your operating system supports additional coded character sets.

In IBM MQ for Windows, ccsid.tbl is located in directory `C:\Program Files\IBM\WebSphere MQ\conv\table` by default.

In IBM MQ for UNIX and Linux systems, ccsid.tbl is located in directory `/var/mqm/conv/table`.

Default data conversion

If you set up channels between two machines on which data conversion is not normally supported, you must enable default data conversion for the channels to work.

To enable default data conversion, edit the ccsid.tbl file to specify a default EBCDIC CCSID and a default ASCII CCSID. Instructions on how to do this are included in the file. You must do this on all machines that will be connected using the channels. Restart the queue manager for the change to take effect.

The default data-conversion process is as follows:

- If conversion between the source and target CCSIDs is not supported, but the CCSIDs of the source and target environments are either both EBCDIC or both ASCII, the character data is passed to the target application without conversion.

- If one CCSID represents an ASCII coded character set, and the other represents an EBCDIC coded character set, IBM MQ converts the data using the default data-conversion CCSIDs defined in ccsid.tbl.

Note: Try to restrict the characters being converted to those that have the same code values in the coded character set specified for the message and in the default coded character set. If you use only the set of characters that is valid for IBM MQ object names (as defined in Naming IBM MQ objects) you will, in general, satisfy this requirement. Exceptions occur with EBCDIC CCSIDs 290, 930, 1279, and 5026 used in Japan, where the lowercase characters have different codes from those used in other EBCDIC CCSIDs.

Converting messages in user-defined formats

The queue manager cannot convert messages in user-defined formats from one coded character set to another. If you need to convert data in a user-defined format, you must supply a data-conversion exit for each such format. Do not use default CCSIDs to convert character data in user-defined formats. For more information about converting data in user-defined formats and about writing data conversion exits, see the Writing data-conversion exits.

Changing the queue manager CCSID

When you have used the CCSID attribute of the ALTER QMGR command to change the CCSID of the queue manager, stop and restart the queue manager to ensure that all running applications, including the command server and channel programs, are stopped and restarted.

This is necessary because any applications that are running when the queue manager CCSID is changed continue to use the existing CCSID.

Administering IBM MQ Telemetry

IBM MQ Telemetry is administered using MQ Explorer or at a command line. Use the explorer to configure telemetry channels, control the telemetry service, and monitor the MQTT clients that are connected to IBM MQ. Configure the security of IBM MQ Telemetry using JAAS, SSL and the IBM MQ object authority manager.

Administering using MQ Explorer

Use the explorer to configure telemetry channels, control the telemetry service, and monitor the MQTT clients that are connected to IBM MQ. Configure the security of IBM MQ Telemetry using JAAS, SSL and the IBM MQ object authority manager.

Administering using the command line

IBM MQ Telemetry can be completely administered at the command line using the IBM MQ MQSC commands.

The IBM MQ Telemetry documentation also has sample scripts that demonstrate the basic usage of the IBM MQ Telemetry Transport v3 Client application.

Read and understand the samples in IBM MQ Telemetry Transport sample programs in the Developing applications for IBM MQ Telemetry section before using them.

Related information:
IBM MQ Telemetry
MQXR properties

Configuring a queue manager for telemetry on Linux and AIX

Follow these manual steps to configure a queue manager to run IBM MQ Telemetry. You can run an automated procedure to set up a simpler configuration using the IBM MQ Telemetry support for MQ Explorer.

Before you begin

1. See Installing IBM MQ Telemetry for information on how to install IBM MQ, and the IBM MQ Telemetry feature.
2. Create and start a queue manager. The queue manager is referred to as *qMgr* in this task.
3. As part of this task you configure the telemetry (MQXR) service. The MQXR property settings are stored in a platform-specific properties file: `mqxr_win.properties` or `mqxr_unix.properties`. You do not normally need to edit the MQXR properties file directly, because almost all settings can be configured through MQSC admin commands or MQ Explorer. If you do decide to edit the file directly, stop the queue manager before you make your changes. See MQXR properties.

About this task

The IBM MQ Telemetry support for MQ Explorer includes a wizard, and a sample command procedure `sampleMQM`. They set up an initial configuration using the guest user ID; see Verifying the installation of IBM MQ Telemetry by using MQ Explorer and IBM MQ Telemetry Transport sample programs.

Follow the steps in this task to configure IBM MQ Telemetry manually using different authorization schemes.

Procedure

1. Open a command window at the telemetry samples directory.

   The telemetry samples directory is `/opt/mqm/mqxr/samples`.
2. Create the telemetry transmission queue.
   ```
   echo "DEFINE QLOCAL('SYSTEM.MQTT.TRANSMIT.QUEUE') USAGE(XMITQ) MAXDEPTH(100000)" | runmqsc qMgr
   ```
   When the telemetry (MQXR) service is first started, it creates `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   It is created manually in this task, because `SYSTEM.MQTT.TRANSMIT.QUEUE` must exist before the telemetry (MQXR) service is started, to authorize access to it.
3. Set the default transmission queue for *qMgr*
   ```
   echo "ALTER QMGR DEFXMITQ('SYSTEM.MQTT.TRANSMIT.QUEUE')" | runmqsc qMgr
   ```
   When the telemetry (MQXR) service is first started, it does not alter the queue manager to make `SYSTEM.MQTT.TRANSMIT.QUEUE` the default transmission queue.

   To make `SYSTEM.MQTT.TRANSMIT.QUEUE` the default transmission queue alter the default transmission queue property. Alter the property using the MQ Explorer or with the command in Figure 19 on page 147.

   Altering the default transmission queue might interfere with your existing configuration. The reason for altering the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE` is to make sending messages directly to MQTT clients easier. Without altering the default transmission queue you must add a remote queue definition for every client that receives IBM MQ messages; see “Sending a message to a client directly” on page 149.
4. Follow a procedure in “Authorizing MQTT clients to access IBM MQ objects” on page 152 to create one or more user IDs. The user IDs have the authority to publish, subscribe, and send publications to MQTT clients.

5. Install the telemetry (MQXR) service

   `cat installMQXRService_unix.mqsc | runmqsc` *`qMgr`*
6. Start the service

   `echo "START SERVICE(SYSTEM.MQXR.SERVICE)" | runmqsc` *`qMgr`*

   The telemetry (MQXR) service is started automatically when the queue manager is started.

   It is started manually in this task, because the queue manager is already running.
7. Using MQ Explorer, configure telemetry channels to accept connections from MQTT clients.
8. Verify the configuration by running the sample client.

   For the sample client to work with your telemetry channel, the channel must authorize the client to publish, subscribe, and receive publications. The sample client connects to the telemetry channel on port 1883 by default.

Example

Figure 18 shows the **`runmqsc`** command to create the `SYSTEM.MQXR.SERVICE` manually on Linux.

```
DEF SERVICE(SYSTEM.MQXR.SERVICE) +
CONTROL(QMGR) +
DESCR('Manages clients using MQXR protocols such as MQTT') +
SERVTYPE(SERVER) +
STARTCMD('+MQ_INSTALL_PATH+/mqxr/bin/runMQXRService.sh') +
STARTARG('-m +QMNAME+ -d "+MQ_Q_MGR_DATA_PATH+" -g "+MQ_DATA_PATH+"') +
STOPCMD('+MQ_INSTALL_PATH+/mqxr/bin/endMQXRService.sh') +
STOPARG('-m +QMNAME+') +
STDOUT('+MQ_Q_MGR_DATA_PATH+/mqxr.stdout') +
STDERR('+MQ_Q_MGR_DATA_PATH+/mqxr.stderr')
```

Figure 18. `installMQXRService_unix.mqsc`

Configuring a queue manager for telemetry on Windows

Follow these manual steps to configure a queue manager to run IBM MQ Telemetry. You can run an automated procedure to set up a simpler configuration using the IBM MQ Telemetry support for MQ Explorer.

Before you begin

1. See Installing IBM MQ Telemetry for information on how to install IBM MQ, and the IBM MQ Telemetry feature.
2. Create and start a queue manager. The queue manager is referred to as *`qMgr`* in this task.
3. As part of this task you configure the telemetry (MQXR) service. The MQXR property settings are stored in a platform-specific properties file: `mqxr_win.properties` or `mqxr_unix.properties`. You do not normally need to edit the MQXR properties file directly, because almost all settings can be configured through MQSC admin commands or MQ Explorer. If you do decide to edit the file directly, stop the queue manager before you make your changes. See MQXR properties.

About this task

The IBM MQ Telemetry support for MQ Explorer includes a wizard, and a sample command procedure `sampleMQM`. They set up an initial configuration using the guest user ID; see Verifying the installation of IBM MQ Telemetry by using MQ Explorer and IBM MQ Telemetry Transport sample programs.

Follow the steps in this task to configure IBM MQ Telemetry manually using different authorization schemes.

Procedure

1. Open a command window at the telemetry samples directory.

   The telemetry samples directory is `WMQ program installation directory\mqxr\samples`.
2. Create the telemetry transmission queue.

   ```
   echo DEFINE QLOCAL('SYSTEM.MQTT.TRANSMIT.QUEUE') USAGE(XMITQ) MAXDEPTH(100000) | runmqsc qMgr
   ```

   When the telemetry (MQXR) service is first started, it creates `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   It is created manually in this task, because `SYSTEM.MQTT.TRANSMIT.QUEUE` must exist before the telemetry (MQXR) service is started, to authorize access to it.
3. Set the default transmission queue for *qMgr*

```
echo ALTER QMGR DEFXMITQ('SYSTEM.MQTT.TRANSMIT.QUEUE') | runmqsc qMgr
```

Figure 19. Set default transmission queue

   When the telemetry (MQXR) service is first started, it does not alter the queue manager to make `SYSTEM.MQTT.TRANSMIT.QUEUE` the default transmission queue.

   To make `SYSTEM.MQTT.TRANSMIT.QUEUE` the default transmission queue alter the default transmission queue property. Alter the property using the MQ Explorer or with the command in Figure 19.

   Altering the default transmission queue might interfere with your existing configuration. The reason for altering the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE` is to make sending messages directly to MQTT clients easier. Without altering the default transmission queue you must add a remote queue definition for every client that receives IBM MQ messages; see "Sending a message to a client directly" on page 149.
4. Follow a procedure in "Authorizing MQTT clients to access IBM MQ objects" on page 152 to create one or more user IDs. The user IDs have the authority to publish, subscribe, and send publications to MQTT clients.
5. Install the telemetry (MQXR) service

   ```
   type installMQXRService_win.mqsc | runmqsc qMgr
   ```
6. Start the service

   ```
   echo START SERVICE(SYSTEM.MQXR.SERVICE) | runmqsc qMgr
   ```

   The telemetry (MQXR) service is started automatically when the queue manager is started.

   It is started manually in this task, because the queue manager is already running.
7. Using MQ Explorer, configure telemetry channels to accept connections from MQTT clients.

   The telemetry channels must be configured such that their identities are one of the user IDs defined in step 4.
8. Verify the configuration by running the sample client.

   For the sample client to work with your telemetry channel, the channel must authorize the client to publish, subscribe, and receive publications. The sample client connects to the telemetry channel on port 1883 by default.

Creating `SYSTEM.MQXR.SERVICE` manually

Figure 20 on page 148 shows the **`runmqsc`** command to create the `SYSTEM.MQXR.SERVICE` manually on Windows.

```
DEF SERVICE(SYSTEM.MQXR.SERVICE) +
CONTROL(QMGR) +
DESCR('Manages clients using MQXR protocols such as MQTT') +
SERVTYPE(SERVER) +
STARTCMD('+MQ_INSTALL_PATH+\mqxr\bin\runMQXRService.bat') +
STARTARG('-m +QMNAME+ -d "+MQ_Q_MGR_DATA_PATH+\." -g "+MQ_DATA_PATH+\."') +
STOPCMD('+MQ_INSTALL_PATH+\mqxr\bin\endMQXRService.bat') +
STOPARG('-m +QMNAME+') +
STDOUT('+MQ_Q_MGR_DATA_PATH+\mqxr.stdout') +
STDERR('+MQ_Q_MGR_DATA_PATH+\mqxr.stderr')
```

Figure 20. `installMQXRService_win.mqsc`

Configure distributed queuing to send messages to MQTT clients

IBM MQ applications can send MQTT v3 clients messages by publishing to subscription created by a client, or by sending a message directly. Whichever method is used, the message is placed on `SYSTEM.MQTT.TRANSMIT.QUEUE`, and sent to the client by the telemetry (MQXR) service. There are a number of ways to place a message on `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Publishing a message in response to an MQTT client subscription

The telemetry (MQXR) service creates a subscription on behalf of the MQTT client. The client is the destination for any publications that match the subscription sent by the client. The telemetry services forwards matching publications back to the client.

An MQTT client is connected to IBM MQ as a queue manager, with its queue manager name set to its `ClientIdentifier`. The destination for publications to be sent to the client is a transmission queue, `SYSTEM.MQTT.TRANSMIT.QUEUE`. The telemetry service forwards messages on `SYSTEM.MQTT.TRANSMIT.QUEUE` to MQTT clients, using the target queue manager name as the key to a specific client.

The telemetry (MQXR) service opens the transmission queue using `ClientIdentifier` as the queue manager name. The telemetry (MQXR) service passes the object handle of the queue to the MQSUB call, to forward publications that match the client subscription. In the object name resolution, the ClientIdentifier is created as the remote queue manager name, and the transmission queue must resolve to `SYSTEM.MQTT.TRANSMIT.QUEUE`. Using standard IBM MQ object name resolution, *`ClientIdentifier`* is resolved as follows; see Table 2 on page 149.

1. *`ClientIdentifier`* matches nothing.

   `ClientIdentifier` is a remote queue manager name. It does not match the local queue manager name, a queue manager alias, or a transmission queue name.

   The queue name is not defined. Currently, the telemetry (MQXR) service sets `SYSTEM.MQTT.PUBLICATION.QUEUE` as the name of the queue. An MQTT v3 client does not support queues, so the resolved queue name is ignored by the client.

   The local queue manager property, `Default transmission queue`, name must be set to `SYSTEM.MQTT.TRANSMIT.QUEUE`, so that the publication is put on `SYSTEM.MQTT.TRANSMIT.QUEUE` to be sent to the client.
2. *`ClientIdentifier`* matches a queue manager alias named *`ClientIdentifier`*.

   `ClientIdentifier` is a remote queue manager name. It matches the name of a queue manager alias.

   The queue manager alias must be defined with *`ClientIdentifier`* as the remote queue manager name.

   By setting the transmission queue name in the queue manager alias definition it is not necessary for the default transmission to be set to `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Table 2. Name resolution of an MQTT queue manager alias

	Input		Output		
`ClientIdentifier`	**Queue manager name**	**Queue name**	**Queue manager name**	**Queue name**	**Transmission queue**
Matches nothing	*`ClientIdentifier`*	*undefined*	*`ClientIdentifier`*	*undefined*	Default transmission queue. `SYSTEM.MQTT.TRANSMIT.QUEUE`
Matches a queue manager alias named *`ClientIdentifier`*	*`ClientIdentifier`*	*undefined*	*`ClientIdentifier`*	*undefined*	`SYSTEM.MQTT.TRANSMIT.QUEUE`

For further information about name resolution, see Name resolution.

Any IBM MQ program can publish to the same topic. The publication is sent to its subscribers, including MQTT v3 clients that have a subscription to the topic.

If an administrative topic is created in a cluster, with the attribute `CLUSTER(clusterName)`, any application in the cluster can publish to the client; for example:

```
echo DEFINE TOPIC('MQTTExamples') TOPICSTR('MQTT Examples') CLUSTER(MQTT) REPLACE | runmqsc qMgr
```

Figure 21. Defining a cluster topic on Windows

Note: Do not give `SYSTEM.MQTT.TRANSMIT.QUEUE` a cluster attribute.

MQTT client subscribers and publishers can connect to different queue managers. The subscribers and publishers can be part of the same cluster, or connected by a publish/subscribe hierarchy. The publication is delivered from the publisher to the subscriber using IBM MQ.

Sending a message to a client directly

An alternative to a client creating a subscription and receiving a publication that matches the subscription topic, send a message to an MQTT v3 client directly. MQTT V3 client applications cannot send messages directly, but other application, such as IBM MQ applications, can.

The IBM MQ application must know the `ClientIdentifier` of the MQTT v3 client. As MQTT v3 clients to not have queues, the target queue name is passed to the MQTT v3 application client messageArrived method as a topic name. For example, in an MQI program, create an object descriptor with the client as the `ObjectQmgrName`:

```
MQOD.ObjectQmgrName = ClientIdentifier ;
MQOD.ObjectName = name ;
```

Figure 22. MQI Object descriptor to send a message to an MQTT v3 client destination

If the application is written using JMS, create a point-to-point destination; for example:

```
javax.jms.Destination jmsDestination =
(javax.jms.Destination)jmsFactory.createQueue
("queue://ClientIdentifier/name");
```

Figure 23. JMS destination to send a message to an MQTT v3 client

To send an unsolicited message to an MQTT client use a remote queue definition. The remote queue manager name must resolvedto the `ClientIdentifier` of the client. The transmission queue must resolve to `SYSTEM.MQTT.TRANSMIT.QUEUE` ; see Table 3. The remote queue name can be anything. The client receives it as a topic string.

Table 3. Name resolution of an MQTT client remote queue definition

Input		Output		
Queue name	**Queue manager name**	**Queue name**	**Queue manager name**	**Transmission queue**
Name of remote queue definition	Blank or local queue manager name	Remote queue name used as a topic string	*`ClientIdentifier`*	`SYSTEM.MQTT.TRANSMIT.QUEUE`

If the client is connected, the message is sent directly to the MQTT client, which calls the messageArrived method; see messageArrived method.

If the client has disconnected with a persistent session, the message is stored in `SYSTEM.MQTT.TRANSMIT.QUEUE` ; see MQTT stateless and stateful sessions. It is forwarded to the client when the client reconnects to the session again.

If you send a non-persistent message it is sent to the client with "at most once" quality of service, `QoS=0`. If you send a persistent message directly to a client, by default, it is sent with "exactly once" quality of service, `QoS=2`. As the client might not have a persistence mechanism, the client can lower the quality of service it accepts for messages sent directly. To lower the quality of service for messages sent directly to a client, make a subscription to the topic `DEFAULT.QoS`. Specify the maximum quality of service the client can support.

MQTT client identification, authorization, and authentication

The telemetry (MQXR) service publishes, or subscribes to, IBM MQ topics on behalf of MQTT clients, using MQTT channels. The IBM MQ administrator configures the MQTT channel identity that is used for IBM MQ authorization. The administrator can define a common identity for the channel, or use the `Username` or `ClientIdentifier` of a client connected to the channel.

The telemetry (MQXR) service can authenticate the client using the `Username` supplied by the client, or by using a client certificate. The `Username` is authenticated using a password provided by the client.

To summarize: Client identification is the selection of the client identity. Depending on the context, the client is identified by the `ClientIdentifier`, `Username`, a common client identity created by the administrator, or a client certificate. The client identifier used for authenticity checking does not have to be the same identifier that is used for authorization.

MQTT client programs set the `Username` and `Password` that are sent to the server using an MQTT channel. They can also set the SSL properties that are required to encrypt and authenticate the connection. The administrator decides whether to authenticate the MQTT channel, and how to authenticate the channel.

To authorize an MQTT client to access IBM MQ objects, authorize the `ClientIdentifier`, or `Username` of the client, or authorize a common client identity. To permit a client to connect to IBM MQ, authenticate the `Username`, or use a client certificate. Configure JAAS to authenticate the `Username`, and configure SSL to authenticate a client certificate.

If you set a `Password` at the client, either encrypt the connection using VPN, or configure the MQTT channel to use SSL, to keep the password private.

It is difficult to manage client certificates. For this reason, if the risks associated with password authentication are acceptable, password authentication is often used to authenticate clients.

If there is a secure way to manage and store the client certificate it is possible to rely on certificate authentication. However, it is rarely the case that certificates can be managed securely in the types of environments that telemetry is used in. Instead, the authentication of devices using client certificates is complemented by authenticating client passwords at the server. Because of the additional complexity, the use of client certificates is restricted to highly sensitive applications. The use of two forms of authentication is called two-factor authentication. You must know one of the factors, such as a password, and have the other, such as a certificate.

In a highly sensitive application, such as a chip-and-pin device, the device is locked down during manufacture to prevent tampering with the internal hardware and software. A trusted, time-limited, client certificate is copied to the device. The device is deployed to the location where it is to be used. Further authentication is performed each time the device is used, either using a password, or another certificate from a smart card.

MQTT client identity and authorization

Use the client ID, `Username`, or a common client identity for authorization to access IBM MQ objects.

The IBM MQ administrator has three choices for selecting the identity of the MQTT channel. The administrator makes the choice when defining or modifying the MQTT channel used by the client. The identity is used to authorize access to IBM MQ topics. The choice is made in the following order:

1. The client ID (see USECLNTID).
2. An identity the administrator provides for the channel (the `MCAUSER` of the channel. See MCAUSER).
3. If neither of the previous choices applies, the `Username` passed from the MQTT client (`Username` is an attribute of the MqttConnectOptions class. It must be set before the client connects to the service. Its default value is null).

Avoid trouble: The identity chosen by this process is thereafter referred to, for example by the DISPLAY CHSTATUS (MQTT) command, as the MCAUSER of the client. Be aware that this is not necessarily the same identity as the MCAUSER of the channel that is referred to in choice (2).

Use the IBM MQ **`setmqaut`** command to select which objects, and which actions, are authorized to be used by the identity associated with the MQTT channel. For example, the following code authorizes a channel identity `MQTTClient`, provided by the administrator of queue manager `QM1`:

```
 setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
```

Authorizing MQTT clients to access IBM MQ objects:

Follow these steps to authorize MQTT clients to publish and subscribe to IBM MQ Objects. The steps follow four alternative access control patterns.

Before you begin

MQTT clients are authorized to access objects in IBM MQ by being assigned an identity when they connect to a telemetry channel. The IBM MQ Administrator configures the telemetry channel using IBM MQ Explorer to give a client one of three types of identity:

1. `ClientIdentifier`
2. `Username`
3. A name the administrator assigns to the channel.

Whichever type is used, the identity must be defined to IBM MQ as a principal by the installed authorization service. The default authorization service on Windows or Linux is called the Object Authority Manager (OAM). If you are using the OAM, the identity must be defined as a user ID.

Use the identity to give a client, or collection of clients, permission to publish or subscribe to topics defined in IBM MQ. If an MQTT client has subscribed to a topic, use the identity to give it permission to receive the resulting publications.

It is hard to manage a system with tens of thousands of MQTT clients, each requiring individual access permissions. One solution is to define common identities, and associate individual MQTT clients with one of the common identities. Define as many common identities as you require to define different combinations of permissions. Another solution is to write your own authorization service that can deal more easily with thousands of users than the operating system.

You can combine MQTT clients into common identities in two ways, using the OAM:

1. Define multiple telemetry channels, each with a different user ID that the administrator allocates using IBM MQ Explorer. Clients connecting using different TCP/IP port numbers are associated with different telemetry channels, and are assigned different identities.

2. Define a single telemetry channel, but have each client select a `Username` from a small set of user IDs. The administrator configures the telemetry channel to select the client `Username` as its identity.

In this task, the identity of the telemetry channel is called *mqttUser*, regardless of how it is set. If collections of clients use different identities, use multiple *mqttUsers*, one for each collection of clients. As the task uses the OAM, each *mqttUser* must be a user ID.

About this task

In this task, you have a choice of four access control patterns that you can tailor to specific requirements. The patterns differ in their granularity of access control.

- "No access control"
- "Coarse-grained access control"
- "Medium-grained access control"
- "Fine-grained access control" on page 154

The result of the models is to assign *mqttUsers* sets of permissions to publish and subscribe to IBM MQ, and receive publications from IBM MQ.

No access control:

MQTT clients are given IBM MQ administrative authority, and can perform any action on any object.

Procedure

1. Create a user ID *mqttUser* to act as the identity of all MQTT clients.
2. Add *mqttUser* to the `mqm` group; see Adding a user to a group on Windows , or Adding a user to a group on Linux

Coarse-grained access control:

MQTT clients have authority to publish and subscribe, and to send messages to MQTT clients. They do not have authority to perform other actions, or to access other objects.

Procedure

1. Create a user ID *mqttUser* to act as the identity of all MQTT clients.
2. Authorize *mqttUser* to publish and subscribe to all topics and to send publications to MQTT clients.
```
setmqaut -m qMgr -t topic -n SYSTEM.BASE.TOPIC -p mqttUser -all +pub +sub
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqttUser -all +put
```

Medium-grained access control:

MQTT clients are divided into different groups to publish and subscribe to different sets of topics, and to send messages to MQTT clients.

Procedure

1. Create multiple user IDs, *mqttUsers*, and multiple administrative topics in the publish/subscribe topic tree.
2. Authorize different *mqttUsers* to different topics.
```
setmqaut -m qMgr -t topic -n topic1 -p mqttUserA -all +pub +sub
setmqaut -m qMgr -t topic -n topic2 -p mqttUserB -all +pub +sub
```
3. Create a group *mqtt*, and add all *mqttUsers* to the group.
4. Authorize *mqtt* to send topics to MQTT clients.
```
setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqtt -all +put
```

Fine-grained access control:

MQTT clients are incorporated into an existing system of access control, that authorizes groups to perform actions on objects.

About this task

A user ID is assigned to one or more operating system groups depending on the authorizations it requires. If IBM MQ applications are publishing and subscribing to the same topic space as MQTT clients, use this model. The groups are referred to as `Publish` *`X`*, `Subscribe` *`Y`*, and `mqtt`

`Publish` *`X`*
: Members of `Publish` *`X`* groups can publish to *`topicX`*.

`Subscribe` *`Y`*
: Members of `Subscribe` *`Y`* groups can subscribe to *`topicY`*.

`mqtt`
: Members of the *`mqtt`* group can send publications to MQTT clients.

Procedure

1. Create multiple groups, `Publish` *`X`* and `Subscribe` *`Y`* that are allocated to multiple administrative topics in the publish/subscribe topic tree.
2. Create a group `mqtt`.
3. Create multiple user IDs, *`mqttUsers`*, and add the users to any of the groups, depending on what they are authorized to do.
4. Authorize different `Publish` *`X`* and `Subscribe` *`X`* groups to different topics, and authorize the *`mqtt`* group to send messages to MQTT clients.

   ```
   setmqaut -m qMgr -t topic -n topic1 -p Publish X -all +pub
   setmqaut -m qMgr -t topic -n topic1 -p Subscribe X -all +pub +sub
   setmqaut -m qMgr -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p mqtt -all +put
   ```

MQTT client authentication using a password

Authenticate the `Username` using the client password. You can authenticate the client using a different identity to the identity used to authorize the client to publish and subscribe to topics.

The telemetry (MQXR) service uses JAAS to authenticate the client `Username`. JAAS uses the `Password` supplied by the MQTT client.

The IBM MQ administrator decides whether to authenticate the `Username`, or not to authenticate at all, by configuring the MQTT channel a client connects to. Clients can be assigned to different channels, and each channel can be configured to authenticate its clients in different ways. Using JAAS, you can configure which methods must authenticate the client, and which can optionally authenticate the client.

The choice of identity for authentication does not affect the choice of identity for authorization. You might want to set up a common identity for authorization for administrative convenience, but authenticate each user to use that identity. The following procedure outlines the steps to authenticate individual users to use a common identity:

1. The IBM MQ administrator sets the MQTT channel identity to any name, such as `MQTTClientUser`, using IBM MQ Explorer.
2. The IBM MQ administrator authorizes `MQTTClient` to publish and subscribe to any topic:

   ```
    setmqaut -m QM1 -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p MQTTClient -all +put
   setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p MQTTClient -all +pub +sub
   ```

3. The MQTT client application developer creates an MqttConnectOptions object and sets `Username` and `Password` before connecting to the server.
4. The security developer creates a JAAS LoginModule to authenticate the `Username` with the `Password` and includes it in the JAAS configuration file.

5. The IBM MQ administrator configures the MQTT channel to authenticate the `UserName` of the client using JAAS.

MQTT client authentication using SSL

Connections between the MQTT client and the queue manager are always initiated by the MQTT client. The MQTT client is always the SSL client. Client authentication of the server and server authentication of the MQTT client are both optional.

By providing the client with a private signed digital certificate, you can authenticate the MQTT client to WebSphere MQ. The WebSphere MQ Administrator can force MQTT clients to authenticate themselves to the queue manager using SSL. You can only request client authentication as part of mutual authentication.

As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

Client authentication using SSL relies upon the client having a secret. The secret is the private key of the client in the case of a self-signed certificate, or a key provided by a certificate authority. The key is used to sign the digital certificate of the client. Anyone in possession of the corresponding public key can verify the digital certificate. Certificates can be trusted, or if they are chained, traced back through a certificate chain to a trusted root certificate. Client verification sends all the certificates in the certificate chain provided by the client to the server. The server checks the certificate chain until it finds a certificate it trusts. The trusted certificate is either the public certificate generated from a self-signed certificate, or a root certificate typically issued by a certificate authority. As a final, optional, step the trusted certificate can be compared with a "live" certificate revocation list.

The trusted certificate might be issued by a certificate authority and already included in the JRE certificate store. It might be a self-signed certificate, or any certificate that has been added to the telemetry channel keystore as a trusted certificate.

Note: The telemetry channel has a combined keystore/truststore that holds both the private keys to one or more telemetry channels, and any public certificates needed to authenticate clients. Because an SSL channel must have a keystore, and it is the same file as the channel truststore, the JRE certificate store is never referenced. The implication is that if authentication of a client requires a CA root certificate, you must place the root certificate in the keystore for the channel, even if the CA root certificate is already in the JRE certificate store. The JRE certificate store is never referenced.

Think about the threats that client authentication is intended to counter, and the roles the client and server play in countering the threats. Authenticating the client certificate alone is insufficient to prevent unauthorized access to a system. If someone else has got hold of the client device, the client device is not necessarily acting with the authority of the certificate holder. Never rely on a single defense against unwanted attacks. At least use a two-factor authentication approach and supplement possession of a certificate with knowledge of private information. For example, use JAAS, and authenticate the client using a password issued by the server.

The primary threat to the client certificate is that it gets into the wrong hands. The certificate is held in a password protected keystore at the client. How does it get placed in the keystore? How does the MQTT client get the password to the keystore? How secure is the password protection? Telemetry devices are often easy to remove, and then can be hacked in private. Must the device hardware be tamper-proof? Distributing and protecting client-side certificates is recognized to be hard; it is called the key-management problem.

A secondary threat is that the device is misused to access servers in unintended ways. For example, if the MQTT application is tampered with, it might be possible to use a weakness in the server configuration using the authenticated client identity.

To authenticate an MQTT client using SSL, configure the telemetry channel, and the client.

Related concepts:

"Telemetry channel configuration for MQTT client authentication using SSL"
The WebSphere MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. The channels are configured either as `com.ibm.mq.MQTT.channel/PlainText` or `com.ibm.mq.MQTT.channel/SSL`. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.

Related information:

MQTT client configuration for client authentication using SSL

Telemetry channel configuration for MQTT client authentication using SSL:

The WebSphere MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. The channels are configured either as `com.ibm.mq.MQTT.channel/PlainText` or `com.ibm.mq.MQTT.channel/SSL`. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.

Set the property, `com.ibm.mq.MQTT.ClientAuth` of an SSL telemetry channel to `REQUIRED` to force all clients connecting on that channel to provide proof that they have verified digital certificates. The client certificates are authenticated using certificates from certificate authorities, leading to a trusted root certificate. If the client certificate is self-signed, or is signed by a certificate that is from a certificate authority, the publicly signed certificates of the client, or certificate authority, must be stored securely at the server.

Place the publicly signed client certificate or the certificate from the certificate authority in the telemetry channel keystore. At the server, publicly signed certificates are stored in the same key file as privately signed certificates, rather than in a separate truststore.

The server verifies the signature of any client certificates it is sent using all the public certificates and cipher suites it has. The server verifies the key chain. The queue manager can be configured to test the certificate against the certificate revocation list. The queue manager revocation namelist property is `SSLCRLNL`.

If any of the certificates a client sends is verified by a certificate in the server keystore, then the client is authenticated.

The IBM MQ administrator can configure the same telemetry channel to use JAAS to check the `UserName` or `ClientIdentifier` of the client with the client `Password`.

You can use the same keystore for multiple telemetry channels.

Verification of at least one digital certificate in the password protected client keystore on the device authenticates the client to the server. The digital certificate is only used for authentication by IBM MQ. It is not used to verify the TCP/IP address of the client, set the identity of the client for authorization or accounting. The identity of the client adopted by the server is either the `Username` or `ClientIdentifier` of the client, or an identity created by the IBM MQ administrator.

You can also use SSL cipher suites for client authentication. If you plan to use SHA-2 cipher suites, see System requirements for using SHA-2 cipher suites with MQTT channels.

Related concepts:

"Telemetry channel configuration for channel authentication using SSL" on page 158
The WebSphere MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. The channels are configured either as `com.ibm.mq.MQTT.channel/PlainText` or `com.ibm.mq.MQTT.channel/SSL`. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.

Related information:

DEFINE CHANNEL (MQTT)

ALTER CHANNEL (MQTT)

CipherSpecs and CipherSuites

Telemetry channel authentication using SSL

Connections between the MQTT client and the queue manager are always initiated by the MQTT client. The MQTT client is always the SSL client. Client authentication of the server and server authentication of the MQTT client are both optional.

The client always attempts to authenticate the server, unless the client is configured to use a CipherSpec that supports anonymous connection. If the authentication fails, then the connection is not established.

As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

Server authentication using SSL authenticates the server to which you are about to send confidential information to. The client performs the checks matching the certificates sent from the server, against certificates placed in its truststore, or in its JRE `cacerts` store.

The JRE certificate store is a JKS file, `cacerts`. It is located in `JRE InstallPath\lib\security\`. It is installed with the default password `changeit`. You can either store certificates you trust in the JRE certificate store, or in the client truststore. You cannot use both stores. Use the client truststore if you want to keep the public certificates the client trusts separate from certificates other Java applications use. Use the JRE certificate store if you want to use a common certificate store for all Java applications running on the client. If you decide to use the JRE certificate store review the certificates it contains, to make sure you trust them.

You can modify the JSSE configuration by supplying a different trust provider. You can customize a trust provider to perform different checks on a certificate. In some OGSi environments that have used the MQTT client, the environment provides a different trust provider.

To authenticate the telemetry channel using SSL, configure the server, and the client.

Telemetry channel configuration for channel authentication using SSL

The WebSphere MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. The channels are configured either as `com.ibm.mq.MQTT.channel/PlainText` or `com.ibm.mq.MQTT.channel/SSL`. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.

Store the digital certificate of the server, signed with its private key, in the keystore that the telemetry channel is going to use at the server. Store any certificates in its key chain in the keystore, if you want to transmit the key chain to the client. Configure the telemetry channel using IBM MQ explorer to use SSL. Provide it with the path to the keystore, and the passphrase to access the keystore. If you do not set the TCP/IP port number of the channel, the SSL telemetry channel port number defaults to 8883.

You can also use SSL cipher suites for channel authentication. If you plan to use SHA-2 cipher suites, see System requirements for using SHA-2 cipher suites with MQTT channels.

Related concepts:

"Telemetry channel configuration for MQTT client authentication using SSL" on page 156
The WebSphere MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. The channels are configured either as `com.ibm.mq.MQTT.channel/PlainText` or `com.ibm.mq.MQTT.channel/SSL`. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.

Related information:

DEFINE CHANNEL (MQTT)

ALTER CHANNEL (MQTT)

CipherSpecs and CipherSuites

Publication privacy on telemetry channels

The privacy of MQTT publications sent in either direction across telemetry channels is secured by using SSL to encrypt transmissions over the connection.

MQTT clients that connect to telemetry channels use SSL to secure the privacy of publications transmitted on the channel using symmetric key cryptography. Because the endpoints are not authenticated, you cannot trust channel encryption alone. Combine securing privacy with server or mutual authentication.

As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

For a typical configuration, which encrypts the channel and authenticates the server, consult "Telemetry channel authentication using SSL" on page 157.

Encrypting SSL connections without authenticating the server exposes the connection to man-in-the-middle attacks. Although the information you exchange is protected against eavesdropping, you do not know who you are exchanging it with. Unless you control the network, you are exposed to someone intercepting your IP transmissions, and masquerading as the endpoint.

You can create an encrypted SSL connection, without authenticating the server, by using a Diffie-Hellman key exchange CipherSpec that supports anonymous SSL. The master secret, shared between the client and server, and used to encrypt SSL transmissions, is established without exchanging a privately signed server certificate.

Because anonymous connections are insecure, most SSL implementations do not default to using anonymous CipherSpecs. If a client request for SSL connection is accepted by a telemetry channel, the channel must have a keystore protected by a passphrase. By default, since SSL implementations do not use anonymous CipherSpecs, the keystore must contain a privately signed certificate that the client can authenticate.

If you use anonymous CipherSpecs, the server keystore must exist, but it need not contain any privately signed certificates.

Another way to establish an encrypted connection is to replace the trust provider at the client with your own implementation. Your trust provider would not authenticate the server certificate, but the connection would be encrypted.

SSL configuration of MQTT Java clients and telemetry channels

Configure SSL to authenticate the telemetry channel and the MQTT Java client, and encrypt the transfer of messages between them. MQTT Java clients use Java Secure Socket Extension (JSSE) to connect telemetry channels using SSL. As an alternative to using SSL, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect MQTT clients to telemetry channels using TCP/IP over the VPN network.

You can configure the connection between a Java MQTT client and a telemetry channel to use the SSL protocol over TCP/IP. What is secured depends on how you configure SSL to use JSSE. Starting with the most secured configuration, you can configure three different levels of security:

1. Permit only trusted MQTT clients to connect. Connect an MQTT client only to a trusted telemetry channel. Encrypt messages between the client and the queue manager; see “MQTT client authentication using SSL” on page 155
2. Connect an MQTT client only to a trusted telemetry channel. Encrypt messages between the client and the queue manager; see “Telemetry channel authentication using SSL” on page 157.
3. Encrypt messages between the client and the queue manager; see “Publication privacy on telemetry channels” on page 158.

JSSE configuration parameters

Modify JSSE parameters to alter the way an SSL connection is configured. The JSSE configuration parameters are arranged into three sets:

1. IBM MQ Telemetry channel
2. MQTT Java client
3. JRE

Configure the telemetry channel parameters using IBM MQ Explorer. Set the MQTT Java Client parameters in the MqttConnectionOptions.SSLProperties attribute. Modify JRE security parameters by editing files in the JRE security directory on both the client and server.

IBM MQ Telemetry channel

Set all the telemetry channel SSL parameters using IBM MQ Explorer.

`ChannelName`

`ChannelName` is a required parameter on all channels.

The channel name identifies the channel associated with a particular port number. Name channels to help you administer sets of MQTT clients.

`PortNumber`

PortNumber is an optional parameter on all channels. It defaults to 1883 for TCP channels, and 8883 for SSL channels.

The TCP/IP port number associated with this channel. MQTT clients are connected to a channel by specifying the port defined for the channel. If the channel has SSL properties, the client must connect using the SSL protocol; for example:

```
MQTTClient mqttClient = new MqttClient( "ssl://www.example.org:8884", "clientId1");
mqttClient.connect();
```

KeyFileName

KeyFileName is a required parameter for SSL channels. It must be omitted for TCP channels.

KeyFileName is the path to the Java keystore containing digital certificates that you provide. Use JKS, JCEKS or PKCS12 as the type of keystore on the server.

Identify the keystore type by using one of the following file extensions:

- .jks
- .jceks
- .p12
- .pkcs12

A keystore with any other file extension is assumed to be a JKS keystore.

You can combine one type of keystore at the server with other types of keystore at the client.

Place the private certificate of the server in the keystore. The certificate is known as the server certificate. The certificate can be self-signed, or part of a certificate chain that is signed by a signing authority.

If you are using a certificate chain, place the associated certificates in the server keystore.

The server certificate, and any certificates in its certificate chain, are sent to clients to authenticate the identity of the server.

If you have set ClientAuth to Required, the keystore must contain any certificates necessary to authenticate the client. The client sends a self-signed certificate, or a certificate chain, and the client is authenticated by the first verification of this material against a certificate in the keystore. Using a certificate chain, one certificate can verify many clients, even if they are issued with different client certificates.

PassPhrase

PassPhrase is a required parameter for SSL channels. It must be omitted for TCP channels.

The passphrase is used to protect the keystore.

ClientAuth

ClientAuth is an optional SSL parameter. It defaults to no client authentication. It must be omitted for TCP channels.

Set ClientAuth if you want the telemetry (MQXR) service to authenticate the client, before permitting the client to connect to the telemetry channel.

If you set ClientAuth, the client must connect to the server using SSL, and authenticate the server. In response to setting ClientAuth, the client sends its digital certificate to the server, and any other certificates in its keystore. Its digital certificate is known as the client certificate. These certificates are authenticated against certificates held in the channel keystore, and in the JRE cacerts store.

CipherSuite

CipherSuite is an optional SSL parameter. It defaults to try all the enabled CipherSpecs. It must be omitted for TCP channels.

If you want to use a particular CipherSpec, set CipherSuite to the name of the CipherSpec that must be used to establish the SSL connection.

The telemetry service and MQTT client negotiate a common CipherSpec from all the CipherSpecs that are enabled at each end. If a specific CipherSpec is specified at either or both ends of the connection, it must match the CipherSpec at the other end.

Install additional ciphers by adding additional providers to JSSE.

Federal Information Processing Standards (FIPS)

FIPS is an optional setting. By default it is not set.

Either in the properties panel of the queue manager, or using **runmqsc**, set SSLFIPS. SSLFIPS specifies whether only FIPS-certified algorithms are to be used.

Revocation namelist

Revocation namelist is an optional setting. By default it is not set.

Either in the properties panel of the queue manager, or using **runmqsc**, set SSLCRLNL. SSLCRLNL specifies a namelist of authentication information objects which are used to provide certificate revocation locations.

No other queue manager parameters that set SSL properties are used.

MQTT Java client

Set SSL properties for the Java client in MqttConnectionOptions.SSLProperties ; for example:

```
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
com.ibm.micro.client.mqttv3.MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setSSLProperties(sslClientProperties);
```

The names and values of specific properties are described in the MqttConnectOptions class. For links to client API documentation for the MQTT client libraries, see MQTT client programming reference.

Protocol

Protocol is optional.

The protocol is selected in negotiation with the telemetry server. If you require a specific protocol you can select one. If the telemetry server does not support the protocol the connection fails.

ContextProvider

ContextProvider is optional.

KeyStore

KeyStore is optional. Configure it if ClientAuth is set at the server to force authentication of the client.

Place the digital certificate of the client, signed using its private key, into the keystore. Specify the keystore path and password. The type and provider are optional. JKS is the default type, and IBMJCE is the default provider.

Specify a different keystore provider to reference a class that adds a new keystore provider. Pass the name of the algorithm used by the keystore provider to instantiate the KeyManagerFactory by setting the key manager name.

TrustStore

TrustStore is optional. You can place all the certificates you trust in the JRE cacerts store.

Configure the truststore if you want to have a different truststore for the client. You might not configure the truststore if the server is using a certificate issued by a well known CA that already has its root certificate stored in cacerts.

Add the publicly signed certificate of the server or the root certificate to the truststore, and specify the truststore path and password. JKS is the default type, and IBMJCE is the default provider.

Specify a different truststore provider to reference a class that adds a new truststore provider. Pass the name of the algorithm used by the truststore provider to instantiate the TrustManagerFactory by setting the trust manager name.

JRE

Other aspects of Java security that affect the behavior of SSL on both the client and server are configured in the JRE. The configuration files on Windows are in *Java Installation Directory*\jre\lib\security. If you are using the JRE shipped with IBM MQ the path is as shown in the following table:

Table 4. Filepaths by platform for JRE SSL configuration files

Platform	Filepath
Windows	*WMQ Installation Directory*\java\jre\lib\security
Linux for System x 32 bit	*WMQ Installation Directory*/java/jre/lib/security
Other UNIX and Linux platforms	*WMQ Installation Directory*/java/jre64/jre/lib/security

Well-known certificate authorities

The cacerts file contains the root certificates of well-known certificate authorities. The cacerts is used by default, unless you specify a truststore. If you use the cacerts store, or do not provide a truststore, you must review and edit the list of signers in cacerts to meet your security requirements.

You can open cacerts using the IBM MQ command strmqikm.which runs the IBM Key Management utility. Open cacerts as a JKS file, using the password changeit. Modify the password to secure the file.

Configuring security classes

Use the java.security file to register additional security providers and other default security properties.

Permissions

Use the java.policy file to modify the permissions granted to resources. javaws.policy grants permissions to javaws.jar

Encryption strength

Some JREs ship with reduced strength encryption. If you cannot import keys into keystores, reduced strength encryption might be the cause. Either, try starting **ikeyman** using the **strmqikm** command, or download strong, but limited jurisdiction files from IBM developer kits, Security information.

Important: Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before downloading or using the unrestricted policy files, you must check the laws of your country. Check its regulations, and its policies concerning the import, possession, use, and re-export of encryption software, to determine if it is permitted.

Modify the trust provider to permit the client to connect to any server

The example illustrates how to add a trust provider and reference it from the MQTT client code. The example performs no authentication of the client or server. The resulting SSL connection is encrypted without being authenticated.

The code snippet in Figure 24 sets the AcceptAllProviders trust provider and trust manager for the MQTT client.

```
java.security.Security.addProvider(new AcceptAllProvider());
java.util.Properties sslClientProperties = new Properties();
sslClientProperties.setProperty("com.ibm.ssl.trustManager","TrustAllCertificates");
sslClientProperties.setProperty("com.ibm.ssl.trustStoreProvider","AcceptAllProvider");
conOptions.setSSLProperties(sslClientProperties);
```

Figure 24. MQTT Client code snippet

```
package com.ibm.mq.id;
public class AcceptAllProvider extends java.security.Provider {
private static final long serialVersionUID = 1L;
public AcceptAllProvider() {
super("AcceptAllProvider", 1.0, "Trust all X509 certificates");
put("TrustManagerFactory.TrustAllCertificates",
AcceptAllTrustManagerFactory.class.getName());
}
```

Figure 25. AcceptAllProvider.java

```
protected static class AcceptAllTrustManagerFactory extends
javax.net.ssl.TrustManagerFactorySpi {
public AcceptAllTrustManagerFactory() {}
protected void engineInit(java.security.KeyStore keystore) {}
protected void engineInit(
javax.net.ssl.ManagerFactoryParameters parameters) {}
protected javax.net.ssl.TrustManager[] engineGetTrustManagers() {
return new javax.net.ssl.TrustManager[] { new AcceptAllX509TrustManager() };
}
```

Figure 26. AcceptAllTrustManagerFactory.java

```
protected static class AcceptAllX509TrustManager implements
javax.net.ssl.X509TrustManager {
public void checkClientTrusted(
java.security.cert.X509Certificate[] certificateChain,
String authType) throws java.security.cert.CertificateException {
report("Client authtype=" + authType);
for (java.security.cert.X509Certificate certificate : certificateChain) {
report("Accepting:" + certificate);
}
}
public void checkServerTrusted(
java.security.cert.X509Certificate[] certificateChain,
String authType) throws java.security.cert.CertificateException {
report("Server authtype=" + authType);
for (java.security.cert.X509Certificate certificate : certificateChain) {
report("Accepting:" + certificate);
}
}
public java.security.cert.X509Certificate[] getAcceptedIssuers() {
return new java.security.cert.X509Certificate[0];
}
private static void report(String string) {
System.out.println(string);
}
}
```

Figure 27. AcceptAllX509TrustManager.java

Telemetry channel JAAS configuration

Configure JAAS to authenticate the `Username` sent by the client.

The IBM MQ administrator configures which MQTT channels require client authentication using JAAS. Specify the name of a JAAS configuration for each channel that is to perform JAAS authentication. Channels can all use the same JAAS configuration, or they can use different JAAS configurations. The configurations are defined in *WMQData directory*\qmgrs*qMgrName*\mqxr\jaas.config.

The `jaas.config` file is organized by JAAS configuration name. Under each configuration name is a list of Login configurations; see Figure 28 on page 165.

JAAS provides four standard Login modules. The standard NT and UNIX Login modules are of limited value.

JndiLoginModule
Authenticates against a directory service configured under JNDI (Java Naming and Directory Interface).

Krb5LoginModule
Authenticates using Kerberos protocols.

NTLoginModule
Authenticates using the NT security information for the current user.

UnixLoginModule
Authenticates using the UNIX security information for the current user.

The problem with using `NTLoginModule` or `UnixLoginModule` is that the telemetry (MQXR) service runs with the `mqm` identity, and not the identity of the MQTT channel. `mqm` is the identity passed to `NTLoginModule` or `UnixLoginModule` for authentication, and not the identity of the client.

To overcome this problem, write your own Login module, or use the other standard Login modules. A sample `JAASLoginModule.java` is supplied with IBM MQ Telemetry. It is an implementation of the `javax.security.auth.spi.LoginModule` interface. Use it to develop your own authentication method.

Any new LoginModule classes you provide must be on the class path of the telemetry (MQXR) service. Do not place your classes in IBM MQ directories that are in the class path. Create your own directories, and define the whole class path for the telemetry (MQXR) service.

You can augment the class path used by the telemetry (MQXR) service by setting class path in the `service.env` file. `CLASSPATH` must be capitalized, and the class path statement can only contain literals. You cannot use variables in the CLASSPATH; for example `CLASSPATH=%CLASSPATH%` is incorrect. The telemetry (MQXR) service sets its own classpath. The CLASSPATH defined in `service.env` is added to it.

The telemetry (MQXR) service provides two callbacks that return the `Username` and the `Password` for a client connected to the MQTT channel. The `Username` and `Password` are set in the MqttConnectOptions object. See Figure 29 on page 166 for an example of how to access `Username` and `Password`.

Examples

An example of a JAAS configuration file with one named configuration, `MQXRConfig`.

```
MQXRConfig {
samples.JAASLoginModule required debug=true;
//com.ibm.security.auth.module.NTLoginModule required;
//com.ibm.security.auth.module.Krb5LoginModule required
//         principal=principal@your_realm
//         useDefaultCcache=TRUE
//         renewTGT=true;
//com.sun.security.auth.module.NTLoginModule required;
//com.sun.security.auth.module.UnixLoginModule required;
//com.sun.security.auth.module.Krb5LoginModule required
//         useTicketCache="true"
//         ticketCache="${user.home}${/}tickets";
};
```

Figure 28. Sample `jaas.config` file

An example of a JAAS Login module coded to receive the `Username` and `Password` provided by an MQTT client.

```
public boolean login()
throws javax.security.auth.login.LoginException {
javax.security.auth.callback.Callback[] callbacks =
new javax.security.auth.callback.Callback[2];
callbacks[0] = new javax.security.auth.callback.NameCallback("NameCallback");
callbacks[1] = new javax.security.auth.callback.PasswordCallback(
"PasswordCallback", false);
try {
callbackHandler.handle(callbacks);
String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
.getName();
char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
.getPassword(); // Accept everything.
if (true) {
loggedIn = true;
} else
throw new javax.security.auth.login.FailedLoginException("Login failed");

principal= new JAASPrincipal(username);

} catch (java.io.IOException exception) {
throw new javax.security.auth.login.LoginException(exception.toString());
} catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
throw new javax.security.auth.login.LoginException(exception.toString());
}

return loggedIn;
}
```

Figure 29. Sample JAASLoginModule.Login() method

Administering IBM MQ Light

V 8.0.0.4

You can administer MQ Light using MQ Explorer or at a command line. Use the Explorer to configure channels and monitor the MQ Light clients that are connected to IBM MQ. Configure the security of MQ Light using TLS and JAAS.

Before you start

For information about installing AMQP on your platform, see Choosing what to install. Install the AMQP Service component by using the IBM MQ V8.0.0.4 manufacturing refresh, not the V8.0.0.4 Fix Pack. You cannot install the AMQP component on a version of the queue manager earlier than V8.0.0.4.

Administering using MQ Explorer

Use the Explorer to configure AMQP channels and monitor the MQ Light clients that are connected to IBM MQ. You can configure the security of MQ Light using TLS and JAAS.

Administering using the command line

You can administer MQ Light at the command line using the IBM MQ MQSC commands.

Viewing IBM MQ objects in use by MQ Light clients

V 8.0.0.4

You can view the different IBM MQ resources in use by MQ Light clients, for example connections and subscriptions.

Connections

When the AMQP service is started new Hconns are created and connected to the queue manager. This pool of Hconns is used when MQ Light clients publish messages. You can view the Hconns by using the **DISPLAY CONN** command. For example:

```
DISPLAY CONN(*) TYPE(CONN) WHERE (APPLDESC LK 'WebSphere MQ Advanced Message Queuing Protocol*')
```

This command also shows any client-specific Hconns. The Hconns that have a blank client ID attribute are the Hconns used in the pool

When an MQ Light client connects to an AMQP channel, a new Hconn is connected to the queue manager. This Hconn is used to consume messages asynchronously for the subscriptions that the MQ Light client has created. You can view the Hconn used by a particular MQ Light client using the **DISPLAY CONN** command. For example:

```
DISPLAY CONN(*) TYPE(CONN) WHERE (CLIENTID EQ 'recv_abcd1234')
```

Subscriptions created by clients

When an MQ Light client subscribes to a topic, a new IBM MQ subscription is created. The subscription name includes the following information:

- The name of the client. If the client joined a shared subscription, the name of the share is used
- The topic pattern that the client subscribed to
- A prefix. The prefix is `private` if the client created a non-shared subscription, or `share` if the client joined a shared subscription

To view the subscriptions in use by a particular MQ Light client, run the **DISPLAY SUB** command and filter on the `private` prefix:

```
DISPLAY SUB(':private:*')
```

To view the shared subscriptions that are in use by multiple clients, run the **DISPLAY SUB** command and filter on the `share` prefix:

```
DISPLAY SUB(':share:*')
```

Because shared subscriptions can be used by multiple MQ Light clients, you might want to view the clients currently consuming messages from the shared subscription. You can do this by listing the Hconns that currently have a handle open on the subscription queue. To view the clients currently using a share, complete the following steps:

1. Find the queue name that the shared subscription uses as a destination. For example:

   ```
   DISPLAY SUB(':private:recv_e298452:public') DEST
      5 : DISPLAY SUB(':private:recv_e298452:public') DEST
   AMQ8096: WebSphere MQ subscription inquired.
      SUBID(414D5120514D31202020202020202020707E0A565C2D0020)
      SUB(:private:recv_e298452:public)
      DEST(SYSTEM.MANAGED.DURABLE.560A7E7020002D5B)
   ```

2. Run the **DISPLAY CONN** command to find the handles open on that queue:

   ```
   DISPLAY CONN(*) TYPE(HANDLE) WHERE  (OBJNAME
   EQ SYSTEM.MANAGED.DURABLE.560A7E7020002D5B)
      21 : DISPLAY CONN(*) TYPE(HANDLE) WHERE(OBJNAME EQ
   ```

```
SYSTEM.MANAGED.DURABLE.560A7E7020002D5B)

AMQ8276: Display Connection details.
   CONN(707E0A56642B0020)
   EXTCONN(414D5143514D3120202020202020202020)
   TYPE(HANDLE)

   OBJNAME(SYSTEM.BASE.TOPIC)      OBJTYPE(TOPIC)

   OBJNAME(SYSTEM.MANAGED.DURABLE.560A7E7020002961)
   OBJTYPE(QUEUE)
```

3. For each of the handles, view the MQ Light client ID that has the handle open:

```
DISPLAY CONN(707E0A56642B0020) CLIENTID
   23 : DISPLAY CONN(707E0A56642B0020) CLIENTID

AMQ8276: Display Connection details.
   CONN(707E0A56642B0020)
   EXTCONN(414D5143514D3120202020202020202020)
   TYPE(CONN)
   CLIENTID(recv_8f02c9d)
DISPLAY CONN(707E0A565F290020) CLIENTID
   24 : DISPLAY CONN(707E0A565F290020) CLIENTID
AMQ8276: Display Connection details.
   CONN(707E0A565F290020)
   EXTCONN(414D5143514D3120202020202020202020)
   TYPE(CONN)
   CLIENTID(recv_86d8888)
```

MQ Light client identification, authorization, and authentication

V 8.0.0.4

Like other IBM MQ client applications, you can secure AMQP connections in a number of ways.

You can use the following security features to secure AMQP connections to IBM MQ:
- Channel authentication records
- Connection authentication
- Channel MCA user configuration
- IBM MQ authority definitions
- TLS connectivity

From a security perspective, establishing a connection consists of the following two steps:
- Deciding whether the connection should continue
- Deciding which IBM MQ identity the application assumes for later authority checks

The following information outlines different IBM MQ configurations and the steps that are worked through when an AMQP client tries to make a connection. Not all IBM MQ configurations use all of the steps described. For example, some configurations do not use TLS for connections inside the company firewall and some configurations use TLS but do not use client certificates for authentication. Many environments do not use custom or custom JAAS modules.

Establishing a connection

The following steps describe what happens when a connection is being established by an AMQP client. The steps determine whether the connection continues and which IBM MQ identity the application assumes for authority checks:

1. If the client opens a TLS connection to IBM MQ and provides a certificate, the queue manager attempts to validate the client certificate.

2. If the client provides user name and password credentials, an AMQP SASL frame is received by the queue manager and MQ CONNAUTH configuration is checked.
3. MQ channel authentication rules are checked (for example, whether the IP address and TLS certificate DN are valid)
4. Channel MCAUSER is asserted, unless channel authentication rules determine otherwise.
5. If a JAAS module has been configured, it is invoked
6. MQ CONNECT authority check applied to resulting MQ user ID.
7. Connection established with an assumed IBM MQ identity.

Publishing a message

The following steps describe what happens when a message is being published by an AMQP client. The steps determine whether the connection continues and which IBM MQ identity the application assumes for authority checks:

1. AMQP link attach frame arrives at queue manager. IBM MQ publish authority for the specified topic string is checked for the MQ user identity established during connection.
2. Message is published to specified topic string.

Subscribing to a topic pattern

The following steps describe what happens when an AMQP client subscribes to a topic pattern. The steps determine whether the connection continues and which IBM MQ identity the application assumes for authority checks:

1. AMQP link attach frame arrives at queue manager. IBM MQ subscribe authority for the specified topic pattern is checked for the MQ user identity established during connection.
2. Subscription is created.

MQ Light client identity and authorization

V 8.0.0.4

Use the MQ Light client ID, the MQ Light user name, or a common client identity defined on the channel or in a channel authentication rule, for authorization to access IBM MQ objects.

The administrator makes the choice when defining or modifying the AMQP channel, by configuring the queue manager CONNAUTH setting, or by defining channel authentication rules. The identity is used to authorize access to IBM MQ topics. The choice is made based on the following:

1. The channel USECLNTID attribute.
2. The ADOPTCTX attribute of the queue manager CONNAUTH rule.
3. The MCAUSER attribute defined on the channel.
4. The USERSRC attribute of a matching channel authentication rule.

Avoid trouble: The identity chosen by this process is thereafter referred to, for example by the DISPLAY CHSTATUS (AMQP) command, as the MCAUSER of the client. Be aware that this is not necessarily the same identity as the MCAUSER of the channel that is referred to in choice (2).

Use the IBM MQ **setmqaut** command to select which objects, and which actions, are authorized to be used by the identity associated with the AMQP channel. For example, the following commands authorize a channel identity `AMQPClient`, provided by the administrator of queue manager `QM1`:

```
setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p AMQPClient -all +pub +sub
```

and

```
setmqaut -m QM1 -t qmgr -p AMQPClient -all +connect
```

MQ Light client authentication using a password

V 8.0.0.4

Authenticate the MQ Light user name using the client password. You can authenticate the client using a different identity from the identity used to authorize the client to publish and subscribe to topics.

The AMQP service can use MQ CONNAUTH or JAAS to authenticate the client user name. If one of these is configured, the password provided by the client is verified by the MQ CONNAUTH configuration or the JAAS module.

The following procedure outlines example steps to authenticate individual users against the local OS users and passwords and, if successful, adopt the common identity `AMQPUser`:

1. The IBM MQ administrator sets the AMQP channel MCAUSER identity to any name, such as `AMQPUser`, using IBM MQ Explorer.
2. The IBM MQ administrator authorizes `AMQPUser` to publish and subscribe to any topic:
   ```
   setmqaut -m QM1 -t topic -n SYSTEM.BASE.TOPIC -p AMQPUser -all +pub +sub +connect
   ```
3. The IBM MQ administrator configures an IDPWOS CONNAUTH rule to check the user name and password presented by the client. The CONNAUTH rule should set CHCKCLNT(REQUIRED) and ADOPTCTX(NO).

Note: You are recommended to use channel authentication rules and to set the MCAUSER channel attribute to a user who has no privileges, to allow more control over connections to the queue manager.

Publication privacy on channels

V 8.0.0.4

The privacy of AMQP publications sent in either direction across AMQP channels is secured by using TLS to encrypt transmissions over the connection.

AMQP clients that connect to AMQP channels use TLS to secure the privacy of publications transmitted on the channel using symmetric key cryptography. Because the endpoints are not authenticated, you cannot trust channel encryption alone. Combine securing privacy with server or mutual authentication.

As an alternative to using TLS, some kinds of Virtual Private Network (VPN), such as IPsec, authenticate the endpoints of a TCP/IP connection. VPN encrypts each IP packet that flows over the network. Once such a VPN connection is established, you have established a trusted network. You can connect AMQP clients to AMQP channels using TCP/IP over the VPN network.

Encrypting TLS connections without authenticating the server exposes the connection to man-in-the-middle attacks. Although the information you exchange is protected against eavesdropping, you do not know who you are exchanging it with. Unless you control the network, you are exposed to someone intercepting your IP transmissions, and masquerading as the endpoint.

You can create an encrypted TLS connection, without authenticating the server, by using a Diffie-Hellman key exchange CipherSpec that supports anonymous TLS. The master secret, shared between the client and server, and used to encrypt TLS transmissions, is established without exchanging a privately signed server certificate.

Because anonymous connections are insecure, most TLS implementations do not default to using anonymous CipherSpecs. If a client request for TLS connection is accepted by an AMQP channel, the channel must have a keystore protected by a passphrase. By default, since TLS implementations do not use anonymous CipherSpecs, the keystore must contain a privately signed certificate that the client can authenticate.

If you use anonymous CipherSpecs, the server keystore must exist, but it need not contain any privately signed certificates.

Another way to establish an encrypted connection is to replace the trust provider at the client with your own implementation. Your trust provider would not authenticate the server certificate, but the connection would be encrypted.

Configuring MQ Light clients with TLS

V 8.0.0.4

You can configure MQ Light clients to use TLS to protect data flowing across the network and to authenticate the identity of the queue manager the client connects to.

To use TLS for the connection from an MQ Light client to an AMQP channel, you must ensure the queue manager has been configured to TLS. Configuring SSL on queue managers describes how to configure the keystore that a queue manager reads TLS certificates from.

When the queue manager has been configured with a keystore, you must configure the TLS attributes on the AMQP channel that clients will connect to. AMQP channels have four attributes related to TLS configuration as follows:

SSLCAUTH
The SSLCAUTH attribute is used to specify whether the queue manager should require an MQ Light client to present a client certificate to verify its identity.

SSLCIPH
The SSLCIPH attribute specifies the cipher the channel should use to encode data in the TLS flow.

SSLPEER
The SSLPEER attribute is used to specify the distinguished name (DN) a client certificate must match if a connection is to be allowed.

CERTLABL
The CERTLABL specifies the certificate the queue manager should present to the client. The queue manager's keystore can contain multiple certificates. This attribute allows you to specify the certificate to be used for connections to this channel. If no CERTLABL is specified the IBM MQ default certificate ibmwebspheremq*qmgr-name* is used.

When you have configured your AMQP channel with the TLS attributes, you must restart the AMQP service using the following command:

```
STOP SERVICE(SYSTEM.AMQP.SERVICE) START SERVICE(SYSTEM.AMQP.SERVICE)
```

When an MQ Light client connects to an AMQP channel protected by TLS, the client verifies the identity of the certificate presented by the queue manager. To do this you must configure your MQ Light client with a truststore containing the queue manager's certificate. The steps to do this vary depending on the MQ Light client you are using.

- For the MQ Light client for Node JS API documentation, see https://www.npmjs.com/package/mqlight
- For the MQ Light client for Java API documentation, see http://mqlight.github.io/java-mqlight/
- For the MQ Light client for Ruby documentation, see http://www.rubydoc.info/github/mqlight/ruby-mqlight/
- For the MQ Light client for Python documentation, see http://python-mqlight.readthedocs.org/en/latest/

Disconnecting MQ Light clients from the queue manager

V 8.0.0.4

If you want to disconnect MQ Light from the queue manager, either run the PURGE CHANNEL command or stop the connection to the MQ Light client.

- Run the **PURGE CHANNEL** command. For example:

  ```
  PURGE CHANNEL(MYAMQP) CLIENTID('recv_28dbb7e')
  ```

- Alternatively, stop the connection that the MQ Light client is using to disconnect the client by completing the following steps:

  1. Find the connection that the client is using by running the **DISPLAY CONN** command. For example:

     ```
     DISPLAY CONN(*) TYPE(CONN) WHERE (CLIENTID EQ 'recv_28dbb7e')
     ```

     The command output is as follows:

     ```
     DISPLAY CONN(*)  TYPE(CONN) WHERE(CLIENTID EQ 'recv_28dbb7e')
       40 : DISPLAY CONN(*) TYPE(CONN) WHERE(CLIENTID EQ 'recv_28dbb7e')
     AMQ8276: Display  Connection details.
       CONN(707E0A565F2D0020)
       EXTCONN(414D5143514D31202020202020202020)
       TYPE(CONN)
       CLIENTID(recv_28dbb7e)
     ```

  2. Stop the connection. For example:

     ```
     STOP CONN(707E0A565F2D0020)
     ```

Administering multicast

Use this information to learn about the IBM MQ Multicast administration tasks such as reducing the size of multicast messages and enabling data conversion.

Getting started with multicast

Use this information to get started with IBM MQ Multicast topics and communication information objects.

About this task

IBM MQ Multicast messaging uses the network to deliver messages by mapping topics to group addresses. The following tasks are a quick way to test if the required IP address and port are correctly configured for multicast messaging.

Creating a COMMINFO object for multicast

The communication information (COMMINFO) object contains the attributes associated with multicast transmission. For more information about the COMMINFO object parameters, see DEFINE COMMINFO.

Use the following command-line example to define a COMMINFO object for multicast:

```
DEFINE COMMINFO(MC1) GRPADDR(group address) PORT(port number)
```

where *MC1* is the name of your COMMINFO object, *group address* is your group multicast IP address or DNS name, and the *port number* is the port to transmit on (The default value is 1414).

A new COMMINFO object called *MC1* is created; This name is the name that you must specify when defining a TOPIC object in the next example.

Creating a TOPIC object for multicast

A topic is the subject of the information that is published in a publish/subscribe message, and a topic is defined by creating a TOPIC object. TOPIC objects have two parameters which define whether they can be used with multicast or not. These parameters are: **COMMINFO** and **MCAST**.

- **COMMINFO** This parameter specifies the name of the multicast communication information object. For more information about the COMMINFO object parameters, see DEFINE COMMINFO.
- **MCAST** This parameter specifies whether multicast is allowable at this position in the topic tree.

Use the following command-line example to define a TOPIC object for multicast:

```
DEFINE TOPIC(ALLSPORTS) TOPICSTR('Sports') COMMINFO(MC1) MCAST(ENABLED)
```

A new TOPIC object called *ALLSPORTS* is created. It has a topic string *Sports*, its related communication information object is called *MC1* (which is the name you specified when defining a COMMINFO object in the previous example), and multicast is enabled.

Testing the multicast publish/subscribe

After the TOPIC and COMMINFO objects have been created, they can be tested using the `amqspubc` sample and the `amqssubc` sample. For more information about these samples see The Publish/Subscribe sample programs.

1. Open two command-line windows; The first command line is for the `amqspubc` publish sample, and the second command line is for the `amqssubc` subscribe sample.
2. Enter the following command at command line 1:
   ```
   amqspubc Sports QM1
   ```
   where *Sports* is the topic string of the TOPIC object defined in an earlier example, and *QM1* is the name of the queue manager.
3. Enter the following command at command line 2:
   ```
   amqssubc Sports QM1
   ```
   where *Sports* and *QM1* are the same as used in step 2.
4. Enter `Hello world` at command line 1. If the port and IP address that are specified in the COMMINFO object are configured correctly; the `amqssubc` sample, which is listening on the port for publications from the specified address, outputs `Hello world` at command line 2.

IBM MQ Multicast topic topology

Use this example to understand the IBM MQ Multicast topic topology.

IBM MQ Multicast support requires that each subtree has its own multicast group and data stream within the total hierarchy.

The *classful network* IP addressing scheme has designated address space for multicast address. The full multicast range of IP address is `224.0.0.0` to `239.255.255.255`, but some of these addresses are reserved. For a list of reserved address either contact your system administrator or see http://www.iana.org/assignments/multicast-addresses for more information. It is recommended that you use the locally scoped multicast address in the range of `239.0.0.0` to `239.255.255.255`.

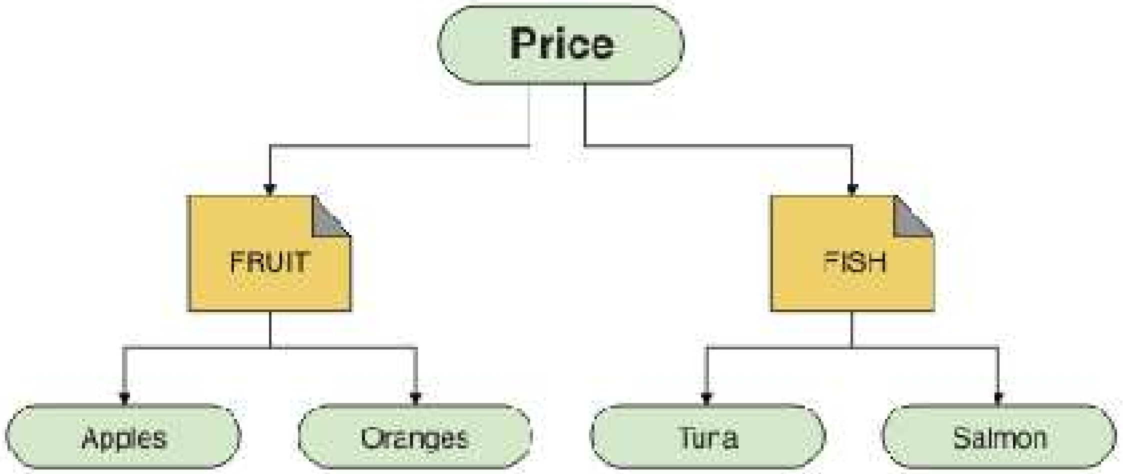
In the following diagram, there are two possible multicast data streams:

```
DEF COMMINFO(MC1) GRPADDR(239.XXX.XXX.XXX
)

DEF COMMINFO(MC2) GRPADDR(239.YYY.YYY.YYY)
```

where *239.XXX.XXX.XXX* and *239.YYY.YYY.YYY* are valid multicast addresses.

These topic definitions are used to create a topic tree as shown in the following diagram:

```
DEFINE TOPIC(FRUIT) TOPICSTRING('Price/FRUIT') MCAST(ENABLED) COMMINFO(MC1)

DEFINE TOPIC(FISH) TOPICSTRING('Price/FISH') MCAST(ENABLED) COMMINFO(MC2)
```



Each multicast communication information (COMMINFO) object represents a different stream of data because their group addresses are different. In this example, the `FRUIT` topic is defined to use COMMINFO object `MC1`, the `FISH` topic is defined to use COMMINFO object `MC2`, and the `Price` node has no multicast definitions.

IBM MQ Multicast has a 255 character limit for topic strings. This limitation means that care must be taken with the names of nodes and leaf-nodes within the tree; if the names of nodes and leaf-nodes are too long, the topic string might exceed 255 characters and return the 2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR reason code. It is recommended to make topic strings as short as possible because longer topic strings might have a detrimental effect on performance.

Controlling the size of multicast messages

Use this information to learn about the IBM MQ message format, and reduce the size of IBM MQ messages.

IBM MQ messages have a number of attributes associated with them which are contained in the message descriptor. For small messages, these attributes might represent most of the data traffic and can have a significant detrimental effect on the transmission rate. IBM MQ Multicast enables the user to configure which, if any, of these attributes are transmitted along with the message.

The presence of message attributes, other than topic string, depends on whether the COMMINFO object states that they must be sent or not. If an attribute is not transmitted, the receiving application applies a default value. The default MQMD values are not necessarily the same as the MQMD_DEFAULT value, and are described later in this topic “Multicast message attributes” on page 175 .

The COMMINFO object contains the MCPROP attribute which controls how many of the MQMD fields and user properties flow with the message. By setting the value of this attribute to an appropriate level, you can control the size of the IBM MQ Multicast messages:

MCPROP
: The multicast properties control how many of the MQMD properties and user properties flow with the message.

  `ALL`
  : All user properties and all the fields of the MQMD are transmitted.

REPLY
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

USER
Only the user properties are transmitted.

NONE
No user properties or MQMD fields are transmitted.

COMPAT
This value causes the transmission of the message to be done in a compatible mode to RMM, which allows some inter-operation with the current XMS applications and IBM Integration Bus RMM applications.

Multicast message attributes

Use this reference information to understand IBM MQ Multicast message attributes.

Message attributes can come from various places, such as the MQMD, the fields in the MQRFH2, and message properties.

The following table shows what happens when messages are sent subject to the value of MCPROP (described previously in this section), and the default value used when an attribute is not sent.

Table 5. Messaging attributes and how they relate to multicast

Attribute	Action when using multicast	Default if not transmitted
TopicString	Always Included	Not applicable
MQMQ StrucId	Not transmitted	Not applicable
MQMD Version	Not transmitted	Not applicable
Report	Included if not default	0
MsgType	Included if not default	MQMT_DATAGRAM
Expiry	Included if not default	0
Feedback	Included if not default	0
Encoding	Included if not default	MQENC_NORMAL(equiv)
CodedCharSetId	Included if not default	1208
Format	Included if not default	MQRFH2
Priority	Included if not default	4
Persistence	Included if not default	MQPER_NOT_PERSISTENT
MsgId	Included if not default	Null
CorrelId	Included if not default	Null
BackoutCount	Included if not default	0
ReplyToQ	Included if not default	Blank
ReplyToQMgr	Included if not default	Blank
UserIdentifier	Included if not default	Blank
AccountingToken	Included if not default	Null

Table 5. Messaging attributes and how they relate to multicast (continued)

Attribute	**Action when using multicast**	**Default if not transmitted**
PutApplType	Included if not default	MQAT_JAVA
PutApplName	Included if not default	Blank
PutDate	Included if not default	Blank
PutTime	Included if not default	Blank
ApplOriginData	Included if not default	Blank
GroupID	Excluded	Not applicable
MsgSeqNumber	Excluded	Not applicable
Offset	Excluded	Not applicable
MsgFlags	Excluded	Not applicable
OriginalLength	Excluded	Not applicable
UserProperties	Included	Not applicable

Related information:
ALTER COMMINFO
DEFINE COMMINFO

Enabling data conversion for Multicast messaging

Use this information to understand how data conversion works for IBM MQ Multicast messaging.

IBM MQ Multicast is a shared, connectionless protocol, and so it is not possible for each client to make specific requests for data conversion. Every client subscribed to the same multicast stream receives the same binary data; therefore, if IBM MQ data conversion is required, the conversion is performed locally at each client.

In a mixed platform installation, it might be that most of the clients require the data in a format that is not the native format of the transmitting application. In this situation the **CCSID** and **ENCODING** values of the multicast COMMINFO object can be used to define the encoding of the message transmission for efficiency.

IBM MQ Multicast supports data conversion of the message payload for the following built in formats:
- MQADMIN
- MQEVENT
- MQPCF
- MQRFH
- MQRFH2
- MQSTR

In addition to these formats, you can also define your own formats and use an MQDXP - Data-conversion exit parameter data conversion exit.

For information about programming data conversions, see Data conversion in the MQI for multicast messaging.

For more information about data conversion, see Data conversion.

For more information about data conversion exits and ClientExitPath, see ClientExitPath stanza of the client configuration file.

Multicast application monitoring

Use this information to learn about administering and monitoring IBM MQ Multicast.

The status of the current publishers and subscribers for multicast traffic (for example, the number of messages sent and received, or the number of messages lost) is periodically transmitted to the server from the client. When status is received, the COMMEV attribute of the COMMINFO object specifies whether or not the queue manager puts an event message on the SYSTEM.ADMIN.PUBSUB.EVENT. The event message contains the status information received. This information is an invaluable diagnostic aid in finding the source of a problem.

Use the MQSC command **DISPLAY CONN** to display connection information about the applications connected to the queue manager. For more information on the **DISPLAY CONN** command, see DISPLAY CONN.

Use the MQSC command **DISPLAY TPSTATUS** to display the status of your publishers and subscribers. For more information on the **DISPLAY TPSTATUS** command, see DISPLAY TPSTATUS.

COMMEV and the multicast message reliability indicator

The *reliability indicator*, used in conjunction with the **COMMEV** attribute of the COMMINFO object, is a key element in the monitoring of IBM MQ Multicast publishers and subscribers. The reliability indicator (the **MSGREL** field that is returned on the Publish or Subscribe status commands) is an IBM MQ indicator that illustrates the percentage of transmissions that have no errors Sometimes messages have to be retransmitted due to a transmission error, which is reflected in the value of **MSGREL**. Potential causes of transmission errors include slow subscribers, busy networks, and network outages. **COMMEV** controls whether event messages are generated for multicast handles that are created using the COMMINFO object and is set to one of three possible values:

DISABLED
Event messages are not written.

ENABLED
Event messages are always written, with a frequency defined in the COMMINFO **MONINT** parameter.

EXCEPTION
Event messages are written if the message reliability is below the reliability threshold. A message reliability level of 90% or less indicates that there might be a problem with the network configuration, or that one or more of the Publish/Subscribe applications is running too slowly:

- A value of **MSGREL(100,100)** indicates that there have been no issues in either the short term, or the long-term time frame.
- A value of **MSGREL(80,60)** indicates that 20% of the messages are currently having issues, but that it is also an improvement on the long-term value of 60.

Clients might continue transmitting and receiving multicast traffic even when the unicast connection to the queue manager is broken, therefore the data might be out of date.

Multicast message reliability

Use this information to learn how to set the IBM MQ Multicast subscription and message history.

A key element of overcoming transmission failure with multicast is the buffering of transmitted data (a history of messages to be kept at the transmitting end of the link) by IBM MQ. This process means that no buffering of messages is required in the putting application process because IBM MQ provides the reliability. The size of this history is configured via the communication information (COMMINFO) object, as described in the following information. A bigger transmission buffer means that there is more transmission history to be retransmitted if needed, but due to the nature of multicast, 100% assured delivery cannot be supported.

The IBM MQ Multicast message history is controlled in the communication information (COMMINFO) object by the **`MSGHIST`** attribute:

MSGHIST

This value is the amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

A value of `0` gives the least level of reliability. The default value is `100` KB.

The IBM MQ Multicast new subscription history is controlled in the communication information (COMMINFO) object by the **`NSUBHIST`** attribute:

NSUBHIST

The new subscriber history controls whether a subscriber joining a publication stream receives as much data as is currently available, or receives only publications made from the time of the subscription.

`NONE`

A value of `NONE` causes the transmitter to transmit only publication made from the time of the subscription. `NONE` is the default value.

`ALL`

A value of `ALL` causes the transmitter to retransmit as much history of the topic as is known. In some circumstances, this situation can give a similar behavior to retained publications.

Note: Using the value of `ALL` might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

Related information:

DEFINE COMMINFO

ALTER COMMINFO

Advanced multicast tasks

Use this information to learn about advanced IBM MQ Multicast administration tasks such as configuring `.ini` files and interoperability with IBM MQ LLM.

For considerations for security in a Multicast installation, see Multicast security.

Bridging between multicast and non-multicast publish/subscribe domains

Use this information to understand what happens when a non-multicast publisher publishes to an IBM MQ Multicast enabled topic.

If a non-multicast publisher publishes to a topic that is defined as **MCAST** enabled and **BRIDGE** enabled, the queue manager transmits the message out over multicast directly to any subscribers that might be listening. A multicast publisher cannot publish to topics that are not multicast enabled.

Existing topics can be multicast enabled by setting the **MCAST** and **COMMINFO** parameters of a topic object. See Initial multicast concepts for more information about these parameters.

The COMMINFO object **BRIDGE** attribute controls publications from applications that are not using multicast. If **BRIDGE** is set to `ENABLED` and the **MCAST** parameter of the topic is also set to `ENABLED`, publications from applications that are not using multicast are bridged to applications that do. For more information on the **BRIDGE** parameter, see DEFINE COMMINFO.

Configuring the .ini files for Multicast

Use this information to understand the IBM MQ Multicast fields in the `.ini` files.

Additional IBM MQ Multicast configuration can be made in an `ini` file. The specific `ini` file that you must use is dependent on the type of applications:

- Client: Configure the *MQ_DATA_PATH* `/mqclient.ini` file.
- Queue manager: Configure the *MQ_DATA_PATH* `/qmgrs/`*QMNAME*`/qm.ini` file.

where *MQ_DATA_PATH* is the location of the IBM MQ data directory (`/var/mqm/mqclient.ini`), and *QMNAME* is the name of the queue manager to which the `.ini` file applies.

The `.ini` file contains fields used to fine-tune the behavior of IBM MQ Multicast:

```
Multicast:
Protocol        = IP | UDP
IPVersion       = IPv4 | IPv6 | ANY | BOTH
LimitTransRate    = DISABLED | STATIC | DYNAMIC
TransRateLimit    = 100000
SocketTTL       = 1
Batch         = NO
Loop        = 1
Interface       = <IPaddress>
FeedbackMode     = ACK | NACK | WAIT1
HeartbeatTimeout   = 20000
HeartbeatInterval   = 2000
```

Protocol

UDP In this mode, packets are sent using the UDP protocol. Network elements cannot provide assistance in the multicast distribution as they do in IP mode however. The packet format remains compatible with PGM. This is the default value.

IP In this mode, the transmitter sends raw IP packets. Network elements with PGM support assist in the reliable multicast packet distribution. This mode is fully compatible with the PGM standard.

IPVersion

IPv4 Communicate using the IPv4 protocol only. This is the default value.

IPv6 Communicate using the IPv6 protocol only.

ANY Communicate using IPv4, IPv6, or both, depending on which protocol is available.

BOTH Supports communication using both IPv4 and IPv6.

LimitTransRate

DISABLED
There is no transmission rate control. This is the default value.

STATIC
Implements static transmission rate control. The transmitter would not transmit at a rate exceeding the rate specified by the TransRateLimit parameter.

DYNAMIC
The transmitter adapts its transmission rate according to the feedback it gets from the receivers. In this case the transmission rate limit cannot be more than the value specified by the TransRateLimit parameter. The transmitter tries to reach an optimal transmission rate.

TransRateLimit
The transmission rate limit in Kbps.

SocketTTL
The value of SocketTTL determines if the multicast traffic can pass through a router, or the number of routers it can pass through.

Batch Controls whether messages are batched or sent immediately There are 2 possible values:

- *NO* The messages are not batched, they are sent immediately.
- *YES* The messages are batched.

Loop Set the value to 1 to enable multicast loop. Multicast loop defines whether the data sent is looped back to the host or not.

Interface
The IP address of the interface on which multicast traffic flows. For more information and troubleshooting, see: Testing multicast applications on a non-multicast network and Setting the appropriate network for multicast traffic

FeedbackMode

NACK
Feedback by negative acknowledgments. This is the default value.

ACK Feedback by positive acknowledgments.

WAIT1
Feedback by positive acknowledgments where the transmitter waits for only 1 ACK from any of the receivers.

HeartbeatTimeout
The heartbeat timeout in milliseconds. A value of 0 indicates that the heartbeat timeout events are not raised by the receiver or receivers of the topic. The default value is 20000.

HeartbeatInterval
The heartbeat interval in milliseconds. A value of 0 indicates that no heartbeats are sent. The heartbeat interval must be considerably smaller than the **`HeartbeatTimeout`** value to avoid false heartbeat timeout events. The default value is 2000.

Multicast interoperability with IBM MQ Low Latency Messaging

Use this information to understand the interoperability between IBM MQ Multicast and IBM MQ Low Latency Messaging (LLM).

Basic payload transfer is possible for an application using LLM, with another application using multicast to exchange messages in both directions. Although multicast uses LLM technology, the LLM product itself is not embedded. Therefore it is possible to install both LLM and IBM MQ Multicast, and operate and service the two products separately.

LLM applications that communicate with multicast might need to send and receive message properties. The IBM MQ message properties and MQMD fields are transmitted as LLM message properties with specific LLM message property codes as shown in the following table:

Table 6. IBM MQ message properties to IBM MQ LLM property mappings

IBM MQ property	IBM MQ LLM property type	LLM property kind	LLM property code
MQMD.Report	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1001
MQMD.MsgType	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1002
MQMD.Expiry	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1003
MQMD.Feedback	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1004
MQMD.Encoding	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1005
MQMD.CodedCharSetId	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1006
MQMD.Format	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1007
MQMD.Priority	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1008
MQMD.Persistence	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1009
MQMD.MsgId	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_ByteArray	-1010
MQMD.BackoutCount	RMM_MSG_PROP_INT32	LLM_PROP_KIND_Int32	-1012
MQMD.ReplyToQ	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1013
MQMD.ReplyToQMger	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1014
MQMD.PutDate	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1020
MQMD.PutTime	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1021
MQMD.ApplOriginData	RMM_MSG_PROP_BYTES	LLM_PROP_KIND_String	-1022
MQPubOptions	RMM_MSG_PROP_INT32	LLM_PROP_KIND_int32	-1053

For more information about LLM, see the LLM product documentation: IBM MQ Low Latency Messaging.

Administering HP Integrity NonStop Server

Use this information to learn about administration tasks for the IBM MQ client for HP Integrity NonStop Server.

Two administration tasks are available to you:

1. Manually starting the TMF/Gateway from Pathway.
2. Stopping the TMF/Gateway from Pathway.

Manually starting the TMF/Gateway from Pathway

You can allow Pathway to automatically start the TMF/Gateway on the first enlistment request, or you can manually start the TMF/Gateway from Pathway.

Procedure

To manually start the TMF/Gateway from Pathway, enter the following PATHCOM command:

```
START SERVER <server_class_name>
```

If a client application makes an enlistment request before the TMF/Gateway completes recovery of in-doubt transactions, the request is held for up to 1 second. If recovery does not complete within that time, the enlistment is rejected. The client then receives an MQRC_UOW_ENLISTMENT_ERROR error from use of a transactional MQI.

Stopping the TMF/Gateway from Pathway

This task describes how to stop the TMF/Gateway from Pathway, and how to restart the TMF/Gateway after you stop it.

Procedure

1. To prevent any new enlistment requests being made to the TMF/Gateway, enter the following command:
   ```
   FREEZE SERVER <server_class_name>
   ```
2. To trigger the TMF/Gateway to complete any in-flight operations and to end, enter the following command:
   ```
   STOP SERVER <server_class_name>
   ```
3. To allow the TMF/Gateway to restart either automatically on first enlistment or manually, following steps 1 and 2, enter the following command:
   ```
   THAW SERVER <server_class_name>
   ```
   Applications are prevented from making new enlistment requests and it is not possible to issue the **START** command until you issue the **THAW** command.

Administering IBM i

IBM i

Introduces the methods available to you to administer IBM MQ on IBM i.

Administration tasks include creating, starting, altering, viewing, stopping, and deleting clusters, processes, and IBM MQ objects (queue managers, queues, namelists, process definitions, channels, client connection channels, listeners, services, and authentication information objects).

See the following links for details of how to administer IBM MQ for IBM i:
- "Managing IBM MQ for IBM i using CL commands"
- "Alternative ways of administering IBM MQ for IBM i" on page 197
- "Work management" on page 202

Related concepts:

"Availability, backup, recovery, and restart" on page 209
Use this information to understand how IBM MQ for IBM i uses the IBM i journaling support to help its backup and restore strategy.

Related reference:

"Quiescing IBM MQ for IBM i" on page 252
This section explains how to quiesce (end gracefully) IBM MQ for IBM i

Related information:

Changing configuration information on IBM i

Understanding IBM MQ for IBM i queue manager library names

Setting up security on IBM i

The dead letter queue handler on IBM i

Determining problems with IBM MQ for IBM i applications

Installable services and components on IBM i

System and default objects on IBM i

Managing IBM MQ for IBM i using CL commands

Use this information to understand the IBM MQ IBM i commands.

Most groups of IBM MQ commands, including those associated with queue managers, queues, topics, channels, namelists, process definitions, and authentication information objects can be accessed using the relevant `WRK*` command.

The principal command in the set is `WRKMQM`. This command allows you, for example, to display a list of all the queue managers on the system, together with status information. Alternatively, you can process all queue-manager specific commands using various options against each entry.

From the `WRKMQM` command you can select specific areas of each queue manager, for example, working with channels, topics or queues, and from there select individual objects.

Recording IBM MQ application definitions

When you create or customize IBM MQ applications, it is useful to keep a record of all IBM MQ definitions created. This record can be used for:
- Recovery purposes
- Maintenance

- Rolling out IBM MQ applications

You can record IBM MQ application definitions in 1 of 2 ways:

1. Creating CL programs to generate your IBM MQ definitions for the server.
2. Creating MQSC text files as SRC members to generate your IBM MQ definitions using the cross-platform IBM MQ command language.

For further details about defining queue objects, see “Script (MQSC) Commands” on page 74 and “Using Programmable Command Formats” on page 7.

Before you start using the IBM MQ for IBM i using CL commands

Use this information to start the IBM MQ subsystem and create a local queue manager.

Before you begin

Ensure that the IBM MQ subsystem is running (using the command `STRSBS QMQM/QMQM`), and that the job queue associated with that subsystem is not held. By default, the IBM MQ subsystem and job queue are both named QMQM in library QMQM.

About this task

Using the IBM i command line to start a queue manager

Procedure

1. Create a local queue manager by issuing the **CRTMQM** command from an IBM i command line. When you create a queue manager, you have the option of making that queue manager the default queue manager. The default queue manager (of which there can only be one) is the queue manager to which a CL command applies, if the queue manager name parameter (MQMNAME) is omitted.
2. Start a local queue manager by issuing the **STRMQM** command from an IBM i command line. If the queue manager startup takes more than a few seconds IBM MQ will show status messages intermittently detailing the start up progress. For more information on these messages see Reason codes.

What to do next

You can stop a queue manager by issuing the **ENDMQM** command from the IBM i command line, and control a queue manager by issuing other IBM MQ commands from an IBM i command line.

Remote queue managers cannot be started remotely but must be created and started in their systems by local operators. An exception to this is where remote operating facilities (outside IBM MQ for IBM i) exist to enable such operations.

The local queue administrator cannot stop a remote queue manager.

Note: As part of quiescing an IBM MQ system, you have to quiesce the active queue managers. This is described in “Quiescing IBM MQ for IBM i” on page 252.

Creating IBM MQ for IBM i objects

Use this information to understand the methods for creating IBM MQ objects for IBM i.

Before you begin

The following tasks suggest various ways in which you can use IBM MQ for IBM i from the command line.

About this task

There are two online methods to create IBM MQ objects, which are:

Procedure

1. Using a Create command, for example: The **`Create MQM Queue`** command: **`CRTMQMQ`**
2. Using a Work with MQM object command, followed by F6, for example: The **`Work with MQM Queues`** command: **`WRKMQMQ`**

What to do next

For a list of all commands see IBM MQ for IBM i CL commands.

Note: All MQM commands can be submitted from the Message Queue Manager Commands menu. To display this menu, type `GO CMDMQM` on the command line and press the `Enter` key.

The system displays the prompt panel automatically when you select a command from this menu. To display the prompt panel for a command that you have typed directly on the command line, press `F4` before pressing the `Enter` key.

Creating a local queue using the CRTMQMQ command:
Procedure

1. Type `CHGMQM` on the command line and press the `F4` key.
2. On the Create MQM Queue panel, type the name of the queue that you want to create in the `Queue name` field. To specify a mixed case name, you enclose the name in apostrophes.
3. Type `*LCL` in the `Queue type` field.
4. Specify a queue manager name, unless you are using the default queue manager, and press the `Enter` key. You can overtype any of the values with a new value. Scroll forward to see further fields. The options used for clusters are at the end of the list of options.
5. When you have changed any values, press the `Enter` key to create the queue.

Creating a local queue using the WRKMQMQ command:
Procedure

1. Type `WRKMQMQ` on the command line.
2. Enter the name of a queue manager.
3. If you want to display the prompt panel, press `F4`. The prompt panel is useful to reduce the number of queues displayed, by specifying a generic queue name or queue type.
4. Press `Enter` and the Work with MQM Queues panel is displayed. You can overtype any of the values with a new value. Scroll forward to see further fields. The options used for clusters are at the end of the list of options.
5. Press `F6` to create a new queue; this takes you to the CRTMQMQ panel. See "Creating a local queue using the CRTMQMQ command" for instructions on how to create the queue. When you have created the queue, the Work with MQM Queues panel is displayed again. The new queue is added to the list when you press `F5=Refresh`.

Altering queue manager attributes:
About this task

To alter the attributes of the queue manager specified on the **CHGMQM** command, specifying the attributes and values that you want to change. For example, use the following options to alter the attributes of `jupiter.queue.manager`:

Procedure

Type **CHGMQM** on the command line and press the `F4` key.

Results

The command changes the dead-letter queue used, and enables inhibit events.

Working with local queues

This section contains examples of some of the commands that you can use to manage local queues. All the commands shown are also available using options from the WRKMQMQ command panel.

Defining a local queue

For an application, the local queue manager is the queue manager to which the application is connected. Queues that are managed by the local queue manager are said to be local to that queue manager.

Use the command **CRTMQMQ QTYPE *LCL** to create a definition of a local queue and also to create the data structure that is called a queue. You can also modify the queue characteristics from those of the default local queue.

In this example, the queue we define, `orange.local.queue`, is specified to have these characteristics:

- It is enabled for gets, disabled for puts, and operates on a first-in-first-out (FIFO) basis.
- It is an *ordinary* queue, that is, it is not an initiation queue or a transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 1000 messages; the maximum message length is 2000 bytes.

The following command does this on the default queue manager:

```
CRTMQMQ QNAME('orange.local.queue') QTYPE(*LCL)
TEXT('Queue for messages from other systems')
PUTENBL(*NO)
GETENBL(*YES)
TRGENBL(*NO)
MSGDLYSEQ(*FIFO)
MAXDEPTH(1000)
MAXMSGLEN(2000)
USAGE(*NORMAL)
```

Note:

1. `USAGE *NORMAL` indicates that this queue is not a transmission queue.
2. If you already have a local queue with the name `orange.local.queue` on the same queue manager, then this command fails. Use the `REPLACE *YES` attribute if you want to overwrite the existing definition of a queue, but see also "Changing local queue attributes" on page 187.

Defining a dead-letter queue

Each queue manager must have a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval. You must explicitly tell the queue manager about the dead-letter queue. You can do this by specifying a dead-letter queue on the **CRTMQM** command, or you can use the **CHGMQM** command to specify one later. You must also define the dead-letter queue before it can be used.

A sample dead-letter queue called `SYSTEM.DEAD.LETTER.QUEUE` is supplied with the product. This queue is automatically created when you create the queue manager. You can modify this definition if required. There is no need to rename it, although you can if you like.

A dead-letter queue has no special requirements except that:
- It must be a local queue.
- Its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle **plus** the size of the dead-letter header (MQDLH).

IBM MQ provides a dead-letter queue handler that allows you to specify how messages found on a dead-letter queue are to be processed or removed. For further information, see The IBM MQ for IBM i dead-letter queue handler.

Displaying default object attributes

When you define an IBM MQ object, it takes any attributes that you do not specify from the default object. For example, when you define a local queue, the queue inherits any attributes that you omit in the definition from the default local queue, which is called `SYSTEM.DEFAULT.LOCAL.QUEUE`. To see exactly what these attributes are, use the following command:

```
DSPMQMQ QNAME(SYSTEM.DEFAULT.LOCAL.QUEUE) MQMNAME(MYQUEUEMANAGER)
```

Copying a local queue definition

You can copy a queue definition using the **CPYMQMQ** command. For example:

```
CPYMQMQ FROMQ('orange.local.queue') TOQ('magenta.queue') MQMNAME(MYQUEUEMANAGER)
```

This command creates a queue with the same attributes as our original queue `orange.local.queue`, rather than those of the system default local queue.

You can also use the **CPYMQMQ** command to copy a queue definition, but substituting one or more changes to the attributes of the original. For example:

```
CPYMQMQ FROMQ('orange.local.queue') TOQ('third.queue') MQMNAME(MYQUEUEMANAGER)
MAXMSGLEN(1024)
```

This command copies the attributes of the queue `orange.local.queue` to the queue `third.queue`, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 2000.

Note: When you use the **CPYMQMQ** command, you copy the queue attributes only, not the messages on the queue.

Changing local queue attributes

You can change queue attributes in two ways, using either the **CHGMQMQ** command or the **CPYMQMQ** command with the REPLACE *YES attribute. In "Defining a local queue" on page 186, you defined the queue `orange.local.queue`. If, for example, you need to increase the maximum message length on this queue to 10,000 bytes.
- Using the **CHGMQMQ** command:

  ```
  CHGMQMQ QNAME('orange.local.queue') MQMNAME(MYQUEUEMANAGER) MAXMSGLEN(10000)
  ```

  This command changes a single attribute, that of the maximum message length; all the other attributes remain the same.

- Using the **CRTMQMQ** command with the REPLACE *YES option, for example:

  ```
  CRTMQMQ QNAME('orange.local.queue') QTYPE(*LCL) MQMNAME(MYQUEUEMANAGER)
  MAXMSGLEN(10000) REPLACE(*YES)
  ```

  This command changes not only the maximum message length, but all the other attributes, which are given their default values. The queue is now put enabled whereas previously it was put inhibited. Put enabled is the default, as specified by the queue `SYSTEM.DEFAULT.LOCAL.QUEUE`, unless you have changed it.

  If you ***decrease*** the maximum message length on an existing queue, existing messages are not affected. Any new messages, however, must meet the new criteria.

Clearing a local queue

To delete all the messages from a local queue called `magenta.queue`, use the following command:

```
CLRMQMQ QNAME('magenta.queue') MQMNAME(MYQUEUEMANAGER)
```

You cannot clear a queue if:
- There are uncommitted messages that have been put on the queue under syncpoint.
- An application currently has the queue open.

Deleting a local queue

Use the command **DLTMQMQ** to delete a local queue.

A queue cannot be deleted if it has uncommitted messages on it, or if it is in use.

Enabling large queues

IBM MQ supports queues larger than 2 GB. See your operating system documentation for information on how to enable IBM i to support large files.

The IBM i product documentation can be found here: http://publib.boulder.ibm.com/iseries/

Some utilities might not be able to cope with files greater than 2 GB. Before enabling large file support, check your operating system documentation for information on restrictions on such support.

Working with alias queues

This section contains examples of some of the commands that you can use to manage alias queues. All the commands shown are also available using options from the WRKMQMQ command panel.

An alias queue (sometimes known as a queue alias) provides a method of redirecting MQI calls. An alias queue is not a real queue but a definition that resolves to a real queue. The alias queue definition contains a target queue name, which is specified by the TGTQNAME attribute.

When an application specifies an alias queue in an MQI call, the queue manager resolves the real queue name at run time.

For example, an application has been developed to put messages on a queue called `my.alias.queue`. It specifies the name of this queue when it makes an **MQOPEN** request and, indirectly, if it puts a message on this queue. The application is not aware that the queue is an alias queue. For each MQI call using this alias, the queue manager resolves the real queue name, which could be either a local queue or a remote queue defined at this queue manager.

By changing the value of the TGTQNAME attribute, you can redirect MQI calls to another queue, possibly on another queue manager. This is useful for maintenance, migration, and load-balancing.

Defining an alias queue

The following command creates an alias queue:

```
CRTMQMQ QNAME('my.alias.queue') QTYPE(*ALS) TGTQNAME('yellow.queue')
MQMNAME(MYQUEUEMANAGER)
```

This command redirects MQI calls that specify `my.alias.queue` to the queue `yellow.queue`. The command does not create the target queue; the MQI calls fail if the queue `yellow.queue` does not exist at run time.

If you change the alias definition, you can redirect the MQI calls to another queue. For example:

```
CHGMQMQ QNAME('my.alias.queue') TGTQNAME('magenta.queue') MQMNAME(MYQUEUEMANAGER)
```

This command redirects MQI calls to another queue, `magenta.queue`.

You can also use alias queues to make a single queue (the target queue) appear to have different attributes for different applications. You do this by defining two aliases, one for each application. Suppose there are two applications:

- Application ALPHA can put messages on `yellow.queue`, but is not allowed to get messages from it.
- Application BETA can get messages from `yellow.queue`, but is not allowed to put messages on it.

You can do this using the following commands:

```
/* This alias is put enabled and get disabled for application ALPHA */

CRTMQMQ QNAME('alphas.alias.queue') QTYPE(*ALS)  TGTQNAME('yellow.queue')
PUTENBL(*YES) GETENBL(*NO) MQMNAME(MYQUEUEMANAGER)

/* This alias is put disabled and get enabled for application BETA */

CRTMQMQ QNAME('betas.alias.queue') QTYPE(*ALS) TGTQNAME('yellow.queue')
PUTENBL(*NO) GETENBL(*YES) MQMNAME(MYQUEUEMANAGER)
```

ALPHA uses the queue name `alphas.alias.queue` in its MQI calls; BETA uses the queue name `betas.alias.queue`. They both access the same queue, but in different ways.

You can use the REPLACE *YES attribute when you define alias queues, in the same way that you use these attributes with local queues.

Using other commands with alias queues

You can use the appropriate commands to display or change alias queue attributes. For example:

```
* Display the alias queue's attributes */

DSPMQMQ QNAME('alphas.alias.queue') MQMNAME(MYQUEUEMANAGER)

/* ALTER the base queue name, to which the alias resolves. */
/* FORCE = Force the change even if the queue is open. */

CHQMQMQ QNAME('alphas.alias.queue') TGTQNAME('orange.local.queue') FORCE(*YES)
MQMNAME(MYQUEUEMANAGER)
```

Working with model queues

This section contains examples of some of the commands that you can use to manage model queues. All the commands shown are also available using options from the WRKMQMQ command panel.

A queue manager creates a dynamic queue if it receives an MQI call from an application specifying a queue name that has been defined as a model queue. The name of the new dynamic queue is generated by the queue manager when the queue is created. A model queue is a template that specifies the attributes of any dynamic queues created from it.

Model queues provide a convenient method for applications to create queues as they are required.

Defining a model queue

You define a model queue with a set of attributes in the same way that you define a local queue. Model queues and local queues have the same set of attributes, except that on model queues you can specify whether the dynamic queues created are temporary or permanent. (Permanent queues are maintained across queue manager restarts, temporary ones are not). For example:

```
CRTMQMQ QNAME('green.model.queue') QTYPE(*MDL) DFNTYPE(*PERMDYN)
```

This command creates a model queue definition. From the DFNTYPE attribute, the actual queues created from this template are permanent dynamic queues. The attributes not specified are automatically copied from the `SYSYTEM.DEFAULT.MODEL.QUEUE` default queue.

You can use the REPLACE *YES attribute when you define model queues, in the same way that you use them with local queues.

Using other commands with model queues

You can use the appropriate commands to display or alter a model queue's attributes. For example:

```
/* Display the model queue's attributes */

DSPMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('green.model.queue')

/* ALTER the model queue to enable puts on any */
/* dynamic queue created from this model. */

CHGMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('blue.model.queue') PUTENBL(*YES)
```

Working with triggering

Use this information to learn about triggering and process definitions.

IBM MQ provides a facility for starting an application automatically when certain conditions on a queue are met. One example of the conditions is when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in Triggering channels.

What is triggering?

The queue manager defines certain conditions as constituting trigger events. If triggering is enabled for a queue and a trigger event occurs, the queue manager sends a trigger message to a queue called an initiation queue. The presence of the trigger message on the initiation queue indicates that a trigger event has occurred.

Trigger messages generated by the queue manager are not persistent. This has the effect of reducing logging (thereby improving performance), and minimizing duplicates during restart, so improving restart time.

What is the trigger monitor?

The program which processes the initiation queue is called a trigger-monitor application, and its function is to read the trigger message and take appropriate action, based on the information contained in the trigger message. Normally this action would be to start some other application to process the queue which caused the trigger message to be generated. From the point of view of the queue manager, there is nothing special about the trigger-monitor application - it is another application that reads messages from a queue (the initiation queue).

Altering the job submission attributes of the trigger monitor

The trigger monitor supplied as command **STRMQMTRM** submits a job for each trigger message using the system default job description, QDFTJOBD. This has limitations in that the submitted jobs are always called QDFTJOBD and have the attributes of the default job description including the library list, *SYSVAL. IBM MQ provides a method for overriding these attributes. For example, it is possible to customize the submitted jobs to have more meaningful job names as follows:

1. In the job description specify the description you want, for example logging values.
2. Specify the Environment Data of the process definition used in the triggering process:
   ```
   CHGMQMPRC PRCNAME(MY_PROCESS) MQMNAME(MHA3) ENVDATA ('JOBD(MYLIB/TRIGJOBD)')
   ```

   The Trigger Monitor performs a SBMJOB using the specified description.

It is possible to override other attributes of the SBMJOB by specifying the appropriate keyword and value in the Environment Data of the process definition. The only exception to this is the CMD keyword because this attribute is filled by the trigger monitor. An example of the command to specify the Environment Data of the process definition where both the job name and description are to be altered follows:

```
CHGMQMPRC PRCNAME(MY_PROCESS) MQMNAME(MHA3) ENVDATA ('JOBD(MYLIB/TRIGJOB)
JOB(TRIGGER)')
```

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the TRGENBL attribute.

In this example, a trigger event is to be generated when there are 100 messages of priority 5 or higher on the local queue motor.insurance.queue, as follows:

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('motor.insurance.queue') QTYPE(*LCL)
PRCNAME('motor.insurance.quote.process') MAXMSGLEN(2000)
DFTMSGPST(*YES) INITQNAME('motor.ins.init.queue')
TRGENBL(*YES) TRGTYPE(*DEPTH) TRGDEPTH(100) TRGMSGPTY(5)
```

where the parameters are:

MQMNAME(MYQUEUEMANAGER)
: The name of the queue manager.

QNAME('motor.insurance.queue')
: The name of the application queue being defined.

PRCNAME('motor.insurance.quote.process')
: The name of the application to be started by a trigger monitor program.

MAXMSGLEN(2000)
: The maximum length of messages on the queue.

DFTMSGPST(*YES)
: Messages on this queue are persistent by default.

INITQNAME('motor.ins.init.queue')
: The name of the initiation queue on which the queue manager is to put the trigger message.

TRGENBL(*YES)
: The trigger attribute value.

TRGTYPE(*DEPTH)
: A trigger event is generated when the number of messages of the required priority (**TRGMSGPTY**) reaches the number specified in **TRGDEPTH**.

TRGDEPTH(100)
: The number of messages required to generate a trigger event.

TRGMSGPTY(5)
: The priority of messages that are to be counted by the queue manager in deciding whether to generate a trigger event. Only messages with priority 5 or higher are counted.

Defining an initiation queue

When a trigger event occurs, the queue manager puts a trigger message on the initiation queue specified in the application queue definition. Initiation queues have no special settings, but you can use the following definition of the local queue `motor.ins.init.queue` for guidance:

```
CRTMQMQ MQMNAME(MYQUEUEMANAGER) QNAME('motor.ins.init.queue') QTYPE(*LCL)
GETENBL(*YES) SHARE(*NO) TRGTYPE(*NONE)
MAXMSGL(2000)
MAXDEPTH(1000)
```

Creating a process definition

Use the **CRTMQMPRC** command to create a process definition. A process definition associates an application queue with the application that is to process messages from the queue. This is done through the PRCNAME attribute on the application queue `motor.insurance.queue`. The following command creates the required process, `motor.insurance.quote.process`, identified in this example:

```
CRTMQMPRC MQMNAME(MYQUEUEMANAGER) PRCNAME('motor.insurance.quote.process')
TEXT('Insurance request message processing')
APPTYPE(*OS400) APPID(MQTEST/TESTPROG)
USRDATA('open, close, 235')
```

where the parameters are:

MQMNAME(MYQUEUEMANAGER)
: The name of the queue manager.

PRCNAME('motor.insurance.quote.process')
: The name of the process definition.

TEXT('Insurance request message processing')
: A description of the application program to which this definition relates. This text is displayed when you use the **DSPMQMPRC** command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotation marks.

APPTYPE(*OS400)
: The type of application to be started.

APPID(MQTEST/TESTPROG)
: The name of the application executable file, specified as a fully qualified file name.

USRDATA('open, close, 235')
: User-defined data, which can be used by the application.

Displaying your process definition

Use the **DSPMQMPRC** command to examine the results of your definition. For example:

```
MQMNAME(MYQUEUEMANAGER) DSPMQMPRC('motor.insurance.quote.process')
```

You can also use the **CHGMQMPRC** command to alter an existing process definition, and the **DLTMQMPRC** command to delete a process definition.

Displaying your process definition

Use the **DSPMQMPRC** command to examine the results of your definition. For example:

```
MQMNAME(MYQUEUEMANAGER) DSPMQMPRC('motor.insurance.quote.process')
```

You can also use the **CHGMQMPRC** command to alter an existing process definition, and the **DLTMQMPRC** command to delete a process definition.

Communicating between two systems

The following example illustrates how to set up two IBM MQ for IBM i systems, using CL commands, so that they can communicate with one another.

The systems are called SYSTEMA and SYSTEMB, and the communications protocol used is TCP/IP.

Carry out the following procedure:

1. Create a queue manager on SYSTEMA, calling it QMGRA1.
   ```
   CRTMQM   MQMNAME(QMGRA1) TEXT('System A - Queue +
   Manager 1') UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   ```
2. Start this queue manager.
   ```
   STRMQM   MQMNAME(QMGRA1)
   ```
3. Define the IBM MQ objects on SYSTEMA that you need to send messages to a queue manager on SYSTEMB.
   ```
   /* Transmission queue */
   CRTMQMQ  QNAME(XMITQ.TO.QMGRB1) QTYPE(*LCL) +
   MQMNAME(QMGRA1) TEXT('Transmission Queue +
   to QMGRB1') MAXDEPTH(5000) USAGE(*TMQ)

   /* Remote queue that points to a queue called TARGETB        */
   /* TARGETB belongs to queue manager QMGRB1 on SYSTEMB        */
   CRTMQMQ  QNAME(TARGETB.ON.QMGRB1) QTYPE(*RMT) +
   MQMNAME(QMGRA1) TEXT('Remote Q pointing +
   at Q TARGETB on QMGRB1 on Remote System +
   SYSTEMB') RMTQNAME(TARGETB) +
   RMTMQMNAME(QMGRB1) TMQNAME(XMITQ.TO.QMGRB1)

   /* TCP/IP sender channel to send messages to the queue manager on SYSTEMB*/
   CRTMQMCHL CHLNAME(QMGRA1.TO.QMGRB1) CHLTYPE(*SDR) +
   MQMNAME(QMGRA1) TRPTYPE(*TCP) +
   TEXT('Sender Channel From QMGRA1 on +
   SYSTEMA to QMGRB1 on SYSTEMB') +
   CONNAME(SYSTEMB) TMQNAME(XMITQ.TO.QMGRB1)
   ```
4. Create a queue manager on SYSTEMB, calling it QMGRB1.
   ```
   CRTMQM   MQMNAME(QMGRB1) TEXT('System B - Queue +
   Manager 1') UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   ```
5. Start the queue manager on SYSTEMB.
   ```
   STRMQM   MQMNAME(QMGRB1)
   ```
6. Define the IBM MQ objects that you need to receive messages from the queue manager on SYSTEMA.

```
/* Local queue to receive messages on */
CRTMQMQ  QNAME(TARGETB) QTYPE(*LCL) MQMNAME(QMGRB1) +
TEXT('Sample Local Queue for QMGRB1')

/* Receiver channel of the same name as the sender channel on SYSTEMA */
CRTMQMCHL CHLNAME(QMGRA1.TO.QMGRB1) CHLTYPE(*RCVR) +
MQMNAME(QMGRB1) TRPTYPE(*TCP) +
TEXT('Receiver Channel from QMGRA1 to +
QMGRB1')
```

7. Finally, start a TCP/IP listener on SYSTEMB so that the channel can be started. This example uses the default port of 1414.

   ```
   STRMQMLSR MQMNAME(QMGRB1)
   ```

You are now ready to send test messages between SYSTEMA and SYSTEMB. Using one of the supplied samples, put a series of messages to your remote queue on SYSTEMA.

Start the channel on SYSTEMA, either by using the command **STRMQMCHL** , or by using the command **WRKMQMCHL** and entering a start request (Option 14) against the sender channel.

The channel should go to RUNNING status and the messages are sent to queue TARGETB on SYSTEMB.

Check your messages by issuing the command:

```
WRKMQMMSG QNAME(TARGETB) MQMNAME(QMGRB1).
```

Sample resource definitions

This sample contains the AMQSAMP4 sample IBM i CL program.

```
/********************************************************************/
/*                              */
/* Program name: AMQSAMP4                     */
/*                              */
/* Description: Sample CL program defining MQM queues        */
/*      to use with the sample programs          */
/*      Can be run, with changes as needed, after      */
/*      starting the MQM                 */
/*                              */
/* <N_OCO_COPYRIGHT>                     */
/* Licensed Materials - Property of IBM              */
/*                              */
/* 63H9336                         */
/* (c) Copyright IBM Corp. 1993, 2005 All Rights Reserved.     */
/*                              */
/* US Government Users Restricted Rights - Use, duplication or   */
/* disclosure restricted by GSA ADP Schedule Contract with     */
/* IBM Corp.                         */
/* <NOC_COPYRIGHT>                      */
/*                              */
/********************************************************************/
/*                              */
/* Function:                         */
/*                              */
/*                              */
/*  AMQSAMP4 is a sample CL program to create or reset the     */
/*  MQI resources to use with the sample programs.        */
/*                              */
/*  This program, or a similar one, can be run when the MQM    */
/*  is started - it creates the objects if missing, or resets   */
/*  their attributes to the prescribed values.          */
/*                              */
/*                              */
/*                              */
/*                              */
/*  Exceptions signaled: none                 */
/*  Exceptions monitored: none                 */
```

```
/*                                 */
/*  AMQSAMP4 takes a single parameter, the Queue Manager name   */
/*                                 */
/*******************************************************************/
QSYS/PGM PARM(&QMGRNAME)

/*******************************************************************/
/*  Queue Manager Name Parameter                 */
/*******************************************************************/
QSYS/DCL VAR(&QMGRNAME) TYPE(*CHAR)

/*******************************************************************/
/*    EXAMPLES OF DIFFERENT QUEUE TYPES             */
/*                                 */
/*  Create local, alias and remote queues             */
/*                                 */
/*  Uses system defaults for most attributes            */
/*                                 */
/*******************************************************************/
/*  Create a local queue                        */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.LOCAL')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)         +
+
TEXT('Sample local queue') /* description */+
SHARE(*YES)        /* Shareable  */+
DFTMSGPST(*YES) /* Persistent messages OK */

/*  Create an alias queue                       */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.ALIAS')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*ALS) REPLACE(*YES)         +
+
TEXT('Sample alias queue')         +
DFTMSGPST(*YES) /* Persistent messages OK */+
TGTQNAME('SYSTEM.SAMPLE.LOCAL')

/*  Create a remote queue - in this case, an indirect reference  */
/*   is made to the sample local queue on OTHER queue manager  */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.REMOTE')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*RMT) REPLACE(*YES)         +
+
TEXT('Sample remote queue')/* description */+
DFTMSGPST(*YES) /* Persistent messages OK */+
RMTQNAME('SYSTEM.SAMPLE.LOCAL')       +
RMTMQMNAME(OTHER)  /* Queue is on OTHER  */

/*  Create a transmission queue for messages to queues at OTHER  */
/*  By default, use remote node name               */
CRTMQMQ   QNAME('OTHER') /* transmission queue name */+
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES) +
TEXT('Transmision queue to OTHER') +
USAGE(*TMQ)  /* transmission queue   */

/*******************************************************************/
/*    SPECIFIC QUEUES AND PROCESS USED BY SAMPLE PROGRAMS    */
/*                                 */
/*  Create local queues used by sample programs          */
/*  Create MQI process associated with sample initiation queue   */
/*                                 */
/*******************************************************************/
/*  General reply queue                        */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.REPLY')        +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)         +
+
TEXT('General reply queue')         +
DFTMSGPST(*NO) /* Not Persistent     */

/*  Queue used by AMQSINQ4                     */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.INQ')         +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)        +
+
TEXT('Queue for AMQSINQ4')         +
SHARE(*YES)        /* Shareable  */+
DFTMSGPST(*NO) /* Not Persistent     */+
+
TRGENBL(*YES) /* Trigger control on   */+
TRGTYPE(*FIRST)/* Trigger on first message*/+
PRCNAME('SYSTEM.SAMPLE.INQPROCESS')     +
INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*  Queue used by AMQSSET4                     */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.SET')         +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)        +
+
TEXT('Queue for AMQSSET4')         +
SHARE(*YES)        /* Shareable */ +
DFTMSGPST(*NO)/* Not Persistent     */ +
+
TRGENBL(*YES) /* Trigger control on   */ +
TRGTYPE(*FIRST)/* Trigger on first message*/+
PRCNAME('SYSTEM.SAMPLE.SETPROCESS')     +
INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*  Queue used by AMQSECH4                      */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.ECHO')         +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)        +
+
TEXT('Queue for AMQSECH4')         +
SHARE(*YES)        /* Shareable */ +
DFTMSGPST(*NO)/* Not Persistent     */ +
+
TRGENBL(*YES) /* Trigger control on   */ +
TRGTYPE(*FIRST)/* Trigger on first message*/+
PRCNAME('SYSTEM.SAMPLE.ECHOPROCESS')    +
INITQNAME('SYSTEM.SAMPLE.TRIGGER')

/*  Initiation Queue used by AMQSTRG4, sample trigger process   */
CRTMQMQ   QNAME('SYSTEM.SAMPLE.TRIGGER') +
MQMNAME(&QMGRNAME)             +
QTYPE(*LCL) REPLACE(*YES)  +
TEXT('Trigger queue for sample programs')

/*  MQI Processes associated with triggered sample programs    */
/*                                 */
/*****  Note - there are versions of the triggered samples ******/
/*****   in different languages - set APPID for these   ******/
/*****   process to the variation you want to trigger   ******/
/*                                 */
CRTMQMPRC  PRCNAME('SYSTEM.SAMPLE.INQPROCESS')     +
MQMNAME(&QMGRNAME)             +
REPLACE(*YES)                +
+
TEXT('Trigger process for AMQSINQ4')    +
ENVDATA('JOBPTY(3)') /* Submit parameter */ +
/** Select the triggered program here       **/        +
APPID('QMQM/AMQSINQ4')  /* C       +
/*    APPID('QMQM/AMQ0INQ4')  /* COBOL */    +
/*    APPID('QMQM/AMQ3INQ4')  /* RPG - ILE */

CRTMQMPRC  PRCNAME('SYSTEM.SAMPLE.SETPROCESS')     +
MQMNAME(&QMGRNAME)            +
REPLACE(*YES)              +
+
TEXT('Trigger process for AMQSSET4')    +
ENVDATA('JOBPTY(3)') /* Submit parameter */ +
/** Select the triggered program here       **/        +
APPID('QMQM/AMQSSET4')  /* C   */    +
/*    APPID('QMQM/AMQ0SET4')  /* COBOL */    +
/*    APPID('QMQM/AMQ3SET4')  /* RPG - ILE */

CRTMQMPRC  PRCNAME('SYSTEM.SAMPLE.ECHOPROCESS')    +
MQMNAME(&QMGRNAME)            +
REPLACE(*YES)              +
+
TEXT('Trigger process for AMQSECH4')    +
ENVDATA('JOBPTY(3)') /* Submit parameter */ +
/** Select the triggered program here       **/        +
APPID('QMQM/AMQSECH4')  /* C   */    +
/*    APPID('QMQM/AMQ0ECH4')  /* COBOL */    +
/*    APPID('QMQM/AMQ3ECH4')  /* RPG - ILE */

/*********************************************************************/
/*                                */
/*  Normal return.                        */
/*                                */
/*********************************************************************/
SNDPGMMSG  MSG('AMQSAMP4 Completed creating sample +
objects for ' *CAT &QMGRNAME)
RETURN
ENDPGM

/*********************************************************************/
/*                                */
/* END OF AMQSAMP4                        */
/*                                */
/*********************************************************************/
```

Alternative ways of administering IBM MQ for IBM i

Use this information to learn about IBM MQ for IBM i, MQSC commands, PCF commands, and remote administration.

You can use IBM MQ instrumentation events to monitor the operation of queue managers. See Instrumentation events for information about IBM MQ instrumentation events and how to use them.

You normally use IBM i CL commands to administer IBM MQ for IBM i. See “Managing IBM MQ for IBM i using CL commands” on page 183 for an overview of these commands.

Using CL commands is the preferred method of administering the system. However, you can use various other methods. This section gives an overview of those methods and includes the following topics:

Local and remote administration

You administer IBM MQ for IBM i objects locally or remotely.

Local administration means carrying out administration tasks on any queue managers that you have defined on your local system. In IBM MQ, you can consider this as local administration because no IBM MQ channels are involved, that is, the communication is managed by the operating system. To perform this type of task, you must either log onto the remote system and issue the commands from there, or create a process that can issue the commands for you.

IBM MQ supports administration from a single point through what is known as *remote administration*. Remote administration consists of sending programmable command format (PCF) control messages to the `SYSTEM.ADMIN.COMMAND.QUEUE` on the target queue manager.

There are a number of ways of generating PCF messages. These are:

1. Writing a program using PCF messages. See "Administration using PCF commands" on page 199.
2. Writing a program using the MQAI, which sends out PCF messages. See "Using the MQAI to simplify the use of PCFs" on page 18.
3. Using the IBM MQ Explorer, available with IBM MQ for Windows, which allows you to use a graphical user interface (GUI) and generates the correct PCF messages. See "Using the IBM MQ Explorer with IBM MQ for IBM i" on page 200.
4. Use **`STRMQMMQSC`** to send commands indirectly to a remote queue manager. See "Administration using MQSC commands."

For example, you can issue a remote command to change a queue definition on a remote queue manager.

Some commands cannot be issued in this way, in particular, creating or starting queue managers and starting command servers. To perform this type of task, you must either log onto the remote system and issue the commands from there or create a process that can issue the commands for you.

Administration using MQSC commands

Use this information to learn about MQSC commands, and how to use them to administer IBM MQ for IBM i.

IBM MQ script (MQSC) commands are written in human-readable form, that is, in EBCDIC text. You use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, namelists, channels, client connection channels, listeners, services, topics, and authentication information objects.

You issue MQSC commands to a queue manager using the **`STRMQMMQSC`** IBM MQ CL command. This method is a batch method only, taking its input from a source physical file in the server library system. The default name for this source physical file is `QMQSC`.

Attention: Do not use the QTEMP library as the source library to STRMQMMQSC, as the usage of the QTEMP library is limited. You must use another library as an input file to the command.

IBM MQ for IBM i does not supply a source file called `QMQSC`. To process MQSC commands you must create the `QMQSC` source file in a library of your choice, by issuing the following command:

```
CRTSRCPF FILE(MYLIB/QMQSC) RCDLEN(240) TEXT('IBM MQ - MQSC Source')
```

MQSC source is held in members within this source file. To work with the members enter the following command:

```
WRKMBRPDM MYLIB/QMQSC
```

You can now add new members and maintain existing ones

You can also enter MQSC commands interactively, by issuing RUNMQSC or:

1. Typing in the queue manager name and pressing the `Enter` key to access the **`WRKMQM`** results panel.
2. Selecting `F23=More options` on this panel.
3. Selecting option 26 against an active queue manager on the panel shown in Figure 30.

To end such an MQSC session, type **end** .

Figure 30 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes.

```
.
.
DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) REPLACE +
DESCR(' ') +
PUT(ENABLED) +
DEFPRTY(0) +
DEFPSIST(NO) +
GET(ENABLED) +
MAXDEPTH(5000) +
MAXMSGL(1024) +
DEFSOPT(SHARED) +
NOHARDENBO +
USAGE(NORMAL) +
NOTRIGGER;
.
.
```

Figure 30. Extract from the MQSC command file, myprog.in

For portability among IBM MQ environments, limit the line length in MQSC command files to 72 characters. The plus sign indicates that the command is continued on the next line.

Object attributes specified in MQSC are shown in this section in uppercase (for example, RQMNAME), although they are not case-sensitive.

Note:

1. The format of an MQSC file does not depend on its location in the file system.
2. MQSC attribute names are limited to eight characters.
3. MQSC commands are available on other platforms, including z/OS.

For a description of each MQSC command and its syntax, see “Script (MQSC) Commands” on page 74.

Administration using PCF commands

The purpose of IBM MQ programmable command format (PCF) commands is to allow administration tasks to be programmed into an administration program. In this way you can create queues and process definitions, and change queue managers, from a program.

PCF commands cover the same range of functions provided by MQSC commands. However, unlike MQSC commands, PCF commands and their replies are not in a text format that you can read.

You can write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of an IBM MQ message. Each command is sent to the target queue manager using the MQI function MQPUT in the same way as any other message. The command server on the queue manager receiving the message interprets it as a command message and runs the command. To get the replies, the application issues an MQGET call and the reply data is returned in another data structure. The application can then process the reply and act accordingly.

Briefly, these are some of the things the application programmer must specify to create a PCF command message:

Message descriptor

This is a standard IBM MQ message descriptor, in which:

- Message type (*MsgType*) is MQMT_REQUEST.
- Message format (*Format*) is MQFMT_ADMIN.

Application data

Contains the PCF message including the PCF header, in which:

- The PCF message type (*Type*) specifies MQCFT_COMMAND.
- The command identifier specifies the command, for example, *Change Queue* (MQCMD_CHANGE_Q).

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. See "Using the MQAI to simplify the use of PCFs" on page 18 for further information.

For a complete description of the PCF data structures and how to implement them, see Structures for commands and responses.

Using the IBM MQ Explorer with IBM MQ for IBM i

Use this information to administer IBM MQ for IBM i using the IBM MQ Explorer.

IBM MQ for Windows (x86 platform), and IBM MQ for Linux (x86 and x86-64 platforms) provide an administration interface called the IBM MQ Explorer to perform administration tasks as an alternative to using CL, control, or MQSC commands.

The IBM MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows (x86 platform), or Linux (x86 and x86-64 platforms), by pointing the IBM MQ Explorer at the queue managers and clusters you are interested in. The platforms and levels of IBM MQ that can be administered using the IBM MQ Explorer are described in Remote queue managers.

With the IBM MQ Explorer, you can:

- Start and stop a queue manager (on your local machine only).
- Define, display, and alter the definitions of IBM MQ objects such as queues, topics, and channels.
- Browse the messages on a queue.
- Start and stop a channel.
- View status information about a channel.
- View queue managers in a cluster.
- Check to see which applications, users, or channels have a particular queue open.
- Create a new queue manager cluster using the Create New Cluster wizard.
- Add a queue manager to a cluster using the Add Queue Manager to Cluster wizard.
- Manage the authentication information object, used with Secure Sockets Layer (SSL) channel security.

Using the online guidance, you can:

- Define and control various resources including queue managers, queues, channels, process definitions, client connection channels, listeners, topics, services, namelists, and clusters.
- Start or stop a queue manager and its associated processes.

- View queue managers and their associated objects on your workstation or from other workstations.
- Check the status of queue managers, clusters, and channels.

Ensure that you have satisfied the following requirements before attempting to use the IBM MQ Explorer to manage IBM MQ on a server machine. Check that:

1. A command server is running for **any** queue manager being administered, started on the server by the CL command **`STRMQMCSVR`**.
2. A suitable TCP/IP listener exists for every remote queue manager. This is the IBM MQ listener started by the **`STRMQMLSR`** command.
3. The server connection channel, called `SYSTEM.ADMIN.SVRCONN`, exists on every remote queue manager. You ***must*** create this channel yourself. It is mandatory for every remote queue manager being administered. Without it, remote administration is not possible.
4. Verify that the `SYSTEM.MQEXPLORER.REPLY.MODEL` queue exists.

Managing the command server for remote administration

Use this information to learn about the remote administration of IBM MQ IBM i command server.

Each queue manager can have a command server associated with it. A command server processes any incoming commands from remote queue managers, or PCF commands from applications. It presents the commands to the queue manager for processing and returns a completion code or operator message depending on the origin of the command.

A command server is mandatory for all administration involving PCFs, the MQAI, and also for remote administration.

Note: For remote administration, you must ensure that the target queue manager is running. Otherwise, the messages containing commands cannot leave the queue manager from which they are issued. Instead, these messages are queued in the local transmission queue that serves the remote queue manager. Avoid this situation if at all possible.

There are separate control commands for starting and stopping the command server. You can perform the operations described in the following sections using the IBM MQ Explorer.

Starting and stopping the command server

To start the command server, use this CL command:

```
STRMQMCSVR MQMNAME('saturn.queue.manager')
```

where `saturn.queue.manager` is the queue manager for which the command server is being started.

To stop the command server, use one of the following CL commands:

1.
   ```
   ENDMQMCSVR MQMNAME('saturn.queue.manager') OPTION(*CNTRLD)
   ```

   to perform a controlled stop, where `saturn.queue.manager` is the queue manager for which the command server is being stopped. This is the default option, which means that the `OPTION(*CNTRLD)` can be omitted.

2.
   ```
   ENDMQMCSVR MQMNAME('saturn.queue.manager') OPTION(*IMMED)
   ```

   to perform an immediate stop, where `saturn.queue.manager` is the queue manager for which the command server is being stopped.

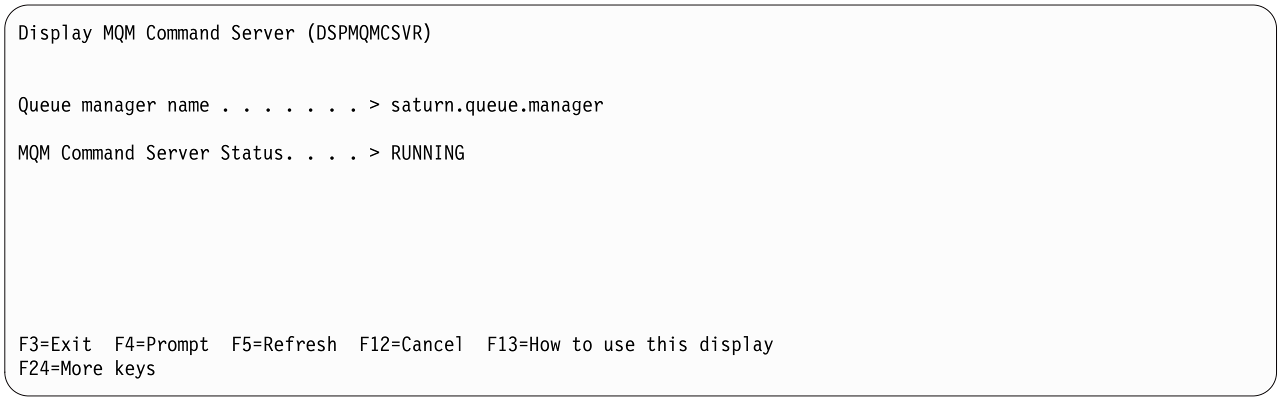
Displaying the status of the command server

For remote administration, ensure that the command server on the target queue manager is running. If it is not running, remote commands cannot be processed. Any messages containing commands are queued in the target queue manager's command queue `SYSTEM.ADMIN.COMMAND.QUEUE`.

To display the status of the command server for a queue manager, called here `saturn.queue.manager`, the CL command is:

```
DSPMQMCSVR MQMNAME('saturn.queue.manager')
```

Issue this command on the target machine. If the command server is running, the panel shown in Figure 31 appears:



```
Display MQM Command Server (DSPMQMCSVR)


Queue manager name . . . . . . . > saturn.queue.manager

MQM Command Server Status. . . . > RUNNING







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

Figure 31. Display MQM Command Server panel

Work management

This information describes the way in which IBM MQ handles work requests, and details the options available for prioritizing and controlling the jobs associated with IBM MQ.

Warning

Do **not** alter IBM MQ work management objects unless you fully understand the concepts of IBM i and IBM MQ work management.

Additional information regarding subsystems and job descriptions can be found under Work Management in the IBM i product documentation. Pay particular attention to the sections on Starting jobs and Batch jobs.

IBM MQ for IBM i incorporates the IBM i UNIX environment and IBM i threads. Do **not** make any changes to the objects in the Integrated File System (IFS).

During normal operations, an IBM MQ queue manager starts a number of batch jobs to perform different tasks. By default these batch jobs run in the QMQM subsystem that is created when IBM MQ is installed.

Work management refers to the process of tailoring IBM MQ tasks to obtain the optimum performance from your system, or to make administration simpler.

For example, you can:

- Change the run-priority of jobs to make one queue manager more responsive than another.
- Redirect the output of a number of jobs to a particular output queue.
- Make all jobs of a certain type run in a specific subsystem.
- Isolate errors to a subsystem.

Work management is carried out by creating or changing the job descriptions associated with the IBM MQ jobs. You can configure work management for:

- An entire IBM MQ installation.
- Individual queue managers.
- Individual jobs for individual queue managers.

Description of IBM MQ tasks

This is a table of the IBM MQ jobs and a brief description of each.

When a queue manager is running, you see some or all of the following batch jobs running under the QMQM user profile in the IBM MQ subsystem. The jobs are described briefly in Table 7.

You can view all jobs connected to a queue manager using option 22 on the Work with Queue Manager (WRKMQM) panel. You can view listeners using the WRKMQMLSR command.

Table 7. IBM MQ tasks.

Job name	**Function**
AMQALMPX	The checkpoint processor that periodically takes journal checkpoints.
AMQZMUC0	Utility manager. This job executes critical queue manager utilities, for example the journal chain manager.
AMQZXMA0	The execution controller that is the first job started by the queue manager. It handles MQCONN requests, and starts agent processes to process IBM MQ API calls.
AMQZFUMA	Object authority manager (OAM).
AMQZLAA0	Queue manager agents that perform most of the work for applications that connect to the queue manager using MQCNO_STANDARD_BINDING.
AMQZLSA0	Queue manager agent.
AMQZMUF0	Utility Manager
AMQZMGR0	Process controller. This job is used to start up and manage listeners and services.
AMQZMUR0	Utility manager. This job executes critical queue manager utilities, for example the journal chain manager.
AMQFQPUB	Queued publish/subscribe daemon.
AMQFCXBA	Broker worker job.
RUNMQBRK	Broker control job.
AMQRMPPA	Channel process pooling job.
AMQCRSTA	TCP/IP-invoked channel responder.
AMQCRS6B	LU62 receiver channel and client connection (see note).
AMQRRMFA	Repository manager for clusters.
AMQCLMAA	Non-threaded TCP/IP listener.
AMQPCSEA	PCF command processor that handles PCF and remote administration requests.
RUNMQTRM	Trigger monitor.
RUNMQDLQ	Dead letter queue handler.
RUNMQCHI	The channel initiator.
RUNMQCHL	Sender channel job that is started for each sender channel.
RUNMQLSR	Threaded TCP/IP listener.
AMQRCMLA	Channel MQSC and PCF command processor.

Note: The LU62 receiver job runs in the communications subsystem and takes its runtime properties from the routing and communications entries that are used to start the job. See Initiated end (Receiver) for more information.

IBM MQ for IBM i work management objects

When IBM MQ is installed, various objects are supplied in the QMQM library to assist with work management. These objects are the ones necessary for IBM MQ jobs to run in their own subsystem.

Sample job descriptions are provided for two of the IBM MQ batch jobs. If no specific job description is provided for an IBM MQ job, it runs with the default job description QMQMJOBD.

The work management objects that are supplied when you install IBM MQ are listed in Table 8 and the objects created for a queue manager are listed in Table 9

Note: The work management objects can be found in the QMQM library and the queue manager objects can be found in the queue manager library.

Table 8. Work management objects

Name	Type	Description
AMQALMPX	*JOBD	The job description that is used by the checkpoint process
AMQZLAA0	*JOBD	The job description that is used by the IBM MQ agent processes
AMQZLSA0	*JOBD	The isolated bindings queue manager agent
AMQZXMA0	*JOBD	The job description that is used by IBM MQ execution controllers
QMQM	*SBSD	The subsystem in which all IBM MQ jobs run
QMQM	*JOBQ	The job queue attached to the supplied subsystem
QMQMJOBD	*JOBD	The default IBM MQ job description, used if there is not a specific job description for a job
QMQMMSG	*MSGQ	The default message queue for IBM MQ jobs.
QMQMRUN20	*CLS	A class description for high priority IBM MQ jobs
QMQMRUN35	*CLS	A class description for medium priority IBM MQ jobs
QMQMRUN50	*CLS	A class description for low priority IBM MQ jobs

Table 9. Work management objects created for a queue manager

Name	Type	Description
AMQA000000	*JRNRCV	Local journal receiver
AMQAJRN	*JRN	Local journal
AMQJRNINF	*USRSPC	User space that is updated with the latest journal receivers required for startup and media recovery of a queue manager. This user space can be queried by an application to determine which journal receivers require archiving and which can be safely deleted.
AMQAJRNMSG	*MSGQ	Local journal message queue
AMQCRC6B	*PGM	Program to start the LU6.2 connection
AMQRFOLD	*FILE	Migrated queue manager channel definition file
QMQMMSG	*MSGQ	Queue manager message queue

How IBM MQ uses the work management objects

This information describes the way in which IBM MQ uses the work management objects, and provides configuration examples.

Attention: Do not alter the job queue entry settings in the QMQM subsytem to limit the number of jobs allowed in the subsystem by priority. If you attempt to do this, you can stop essential IBM MQ jobs from running after they are submitted and cause the queue manager startup to fail.

To understand how to configure work management, you must first understand how IBM MQ uses job descriptions.

The job description used to start the job controls many attributes of the job. For example:

- The job queue on which the job is queued and on which subsystem the job runs.
- The routing data used to start the job and class that the job uses for its runtime parameters.
- The output queue that the job uses for print files.

The process of starting an IBM MQ job can be considered in three steps:

1. IBM MQ selects a job description.

   IBM MQ uses the following technique to determine which job description to use for a batch job:

   a. Look in the queue manager library for a job description with the same name as the job. See Understanding IBM MQ for IBM i queue manager library names for further details about the queue manager library.

   b. Look in the queue manager library for the default job description QMQMJOBD.

   c. Look in the QMQM library for a job description with the same name as the job.

   d. Use the default job description, QMQMJOBD, in the QMQM library.

2. The job is submitted to the job queue.

   Job descriptions supplied with IBM MQ have been set up, by default, to put jobs on to job queue QMQM in library QMQM. The QMQM job queue is attached to the supplied QMQM subsystem, so by default the jobs start running in the QMQM subsystem.

3. The job enters the subsystem and goes through the routing steps.

   When the job enters the subsystem, the routing data specified on the job description is used to find routing entries for the job.

   The routing data must match one of the routing entries defined in the QMQM subsystem, and this defines which of the supplied classes (QMQMRUN20, QMQMRUN35, or QMQMRUN50) is used by the job.

Note: If IBM MQ jobs do not appear to be starting, make sure that the subsystem is running and the job queue is not held,

If you have modified the IBM MQ work management objects, make sure everything is associated correctly. For example, if you specify a job queue other than QMQM/QMQM on the job description, make sure that an ADDJOBQE is performed for the subsystem, that is, QMQM.

You can create a job description for each job documented in Table 7 on page 203 using the following worksheet as an example:

```
What is the queue manager library name? __________
Does job description AMQZXMA0 exist in the queue manager library? Yes   No
Does job description QMQMJOBD exist in the queue manager library? Yes   No
Does job description AMQZXMA0 exist in the QMQM library?     Yes   No
Does job description QMQMJOBD exist in the QMQM library?     Yes   No
```

If you answer No to all these questions, create a global job description QMQMJOBD in the QMQM library.

The IBM MQ message queue

An IBM MQ message queue, QMQMMSG, is created in each queue manager library. Operating system messages are sent to this queue when queue manager jobs end and IBM MQ sends messages to the queue. For example, to report which journal receivers are needed at startup. Keep the number of messages in this message queue at a manageable size to make it easier to monitor.

Default system examples

These examples show how an unmodified IBM MQ installation works when some of the standard jobs are submitted at queue manager startup time.

First, the AMQZXMA0 execution controller job starts.

1. Issue the **STRMQM** command for queue manager `TESTQM`.
2. IBM MQ searches the queue manager library QMTESTQM, firstly for job description AMQZXMA0, and then job description QMQMJOBD.

   Neither of these job descriptions exist, so IBM MQ looks for job description AMQZXMA0 in the product library QMQM. This job description exists, so it is used to submit the job.
3. The job description uses the IBM MQ default job queue, so the job is submitted to job queue QMQM/QMQM.
4. The routing data on the AMQZXMA0 job description is QMQMRUN20, so the system searches the subsystem routing entries for one that matches that data.

   By default, the routing entry with sequence number 9900 has comparison data that matches QMQMRUN20, so the job is started with the class defined on that routing entry, which is also called QMQMRUN20.
5. The QMQM/QMQMRUN20 class has run priority set to 20, so the AMQZXMA0 job runs in subsystem QMQM with the same priority as most interactive jobs on the system.

Next, the AMQALMPX checkpoint process job starts.

1. IBM MQ searches the queue manager library QMTESTQM, firstly for job description AMQALPMX, and then job description QMQMJOBD.

   Neither of these job descriptions exist, so IBM MQ looks for job descriptions AMQALMPX and QMQMJOBD in the product library QMQM.

   Job description AMQALMPX does not exist but QMQMJOBD does, so QMQMJOBD is used to submit the job.

   Note: The QMQMJOBD job description is always used for IBM MQ jobs that do not have their own job description.
2. The job description uses the IBM MQ default job queue, so the job is submitted to job queue QMQM/QMQM.
3. The routing data on the QMQMJOBD job description is QMQMRUN35, so the system searches the subsystem routing entries for one that matches that data.

   By default, the routing entry with sequence number 9910 has comparison data that matches QMQMRUN35, so the job is started with the class defined on that routing entry, which is also called QMQMRUN35.
4. The QMQM/QMQMRUN35 class has run priority set to 35, so the AMQALMPX job runs in subsystem QMQM with a lower priority than most interactive jobs on the system, but higher priority than most batch jobs.

Configuring work management examples

Use this information to learn how you can change and create IBM MQ job descriptions to change the runtime attributes of IBM MQ jobs.

The key to the flexibility of IBM MQ work management lies in the two-tier way that IBM MQ searches for job descriptions:

- If you create or change job descriptions in a queue manager library, those changes override the global job descriptions in QMQM, but the changes are *local* and affect that particular queue manager alone.
- If you create or change global job descriptions in the QMQM library, those job descriptions affect all queue managers on the system, unless overridden locally for individual queue managers.

1. The following example increases the priority of channel control jobs for an individual queue manager.

   To make the repository manager and channel initiator jobs, AMQRRMFA and RUNMQCHI, run as quickly as possible for queue manager TESTQM, carry out the following steps:

   a. Create local duplicates of the QMQM/QMQMJOBD job description with the names of the IBM MQ processes that you want to control in the queue manager library. For example:

   ```
   CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
   NEWOBJ(RUNMQCHI)
   CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
   NEWOBJ(AMQRRMFA)
   ```

   b. Change the routing data parameter on the job description to ensure that the jobs use the QMQMRUN20 class.

   ```
   CHGJOBD JOBD(QMTESTQM/RUNMQCHI) RTGDTA('QMQMRUN20')
   CHGJOBD JOBD(QMTESTQM/AMQRRMFA) RTGDTA('QMQMRUN20')
   ```

   The AMQRRMFA and RUNMQCHI jobs for queue manager TESTQM now:

   - Use the new local job descriptions in the queue manager library
   - Run with priority 20, because the QMQMRUN20 class is used when the jobs enter the subsystem.

2. The following example defines a new run priority class for the QMQM subsystem.

   a. Create a duplicate class in the QMQM library, to allow other queue managers to access the class, by issuing the following command:

   ```
   CRTDUPOBJ OBJ(QMQMRUN20) FROMLIB(QMQM) OBJTYPE(*CLS) TOLIB(QMQM)
   NEWOBJ(QMQMRUN10)
   ```

   b. Change the class to have the new run priority by issuing the following command:

   ```
   CHGCLS CLS(QMQM/QMQMRUN10) RUNPTY(10)
   ```

   c. Add the new class definition to the subsystem by issuing the following command:

   ```
   ADDRTGE SBSD(QMQM/QMQM) SEQNBR(8999) CMPVAL('QMQMRUN10') PGM(QSYS/QCMD)
   CLS(QMQM/QMQMRUN10)
   ```

   Note: You can specify any numeric value for the routing sequence number, but the values must be in sequential order. This sequence number tells the subsystem the order in which routing entries are to be searched for a routing data match.

   d. Change the local or global job description to use the new priority class by issuing the following command:

   ```
   CHGJOBD JOBD(QMQMlibname/QMQMJOBD) RTGDTA('QMQMRUN10')
   ```

   Now all the queue manager jobs associated with the QMlibraryname use a run priority of 10.

3. The following example runs a queue manager in its own subsystem

   To make all the jobs for queue manager TESTQM run in the QBATCH subsystem, carry out the following steps:

   a. Create a local duplicate of the QMQM/QMQMJOBD job description in the queue manager library with the command

   ```
   CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMTESTQM)
   ```

b. Change the job queue parameter on the job description to ensure that the jobs use the QBATCH job queue.

```
CHGJOBD JOBD(QMTESTQM/QMQMJOBD) JOBQ(*LIBL/QBATCH)
```

Note: The job queue is associated with the subsystem description. If you find that the jobs are staying on the job queue, verify that the job queue definition is defined on the SBSD. Use the DSPSBSD command for the subsystem and take option **6**, "Job queue entries".

All jobs for queue manager TESTQM now:

- Use the new local default job description in the queue manager library
- Are submitted to job queue QBATCH.

To ensure that jobs are routed and prioritized correctly:

- Either create routing entries for the IBM MQ jobs in subsystem QBATCH, or
- Rely on a catch-all routing entry that calls QCMD, irrespective of what routing data is used.

  This option works only if the maximum active jobs option for job queue QBATCH is set to *NOMAX. The system default is 1.

4. The following example creates another IBM MQ subsystem
   a. Create a duplicate subsystem in the QMQM library by issuing the following command:
   ```
   CRTDUPOBJ OBJ(QMQM) FROMLIB(QMQM) OBJTYPE(*SBSD) TOLIB(QMQM) NEWOBJ(QMQM2)
   ```
   b. Remove the QMQM job queue by issuing the following command:
   ```
   RMVJOBQE SBSD(QMQM/QMQM2) JOBQ(QMQM/QMQM)
   ```
   c. Create a new job queue for the subsystem by issuing the following command:
   ```
   CRTJOBQ JOBQ(QMQM/QMQM2) TEXT('Job queue for IBM MQ Queue Manager')
   ```
   d. Add a job queue entry to the subsystem by issuing the following command:
   ```
   ADDJOBQE SBSD(QMQM/QMQM2) JOBQ(QMQM/QMQM2) MAXACT(*NOMAX)
   ```
   e. Create a duplicate QMQMJOBD in the queue manager library by issuing the following command:
   ```
   CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) TOLIB(QMlibraryname)
   ```
   f. Change the job description to use the new job queue by issuing the following command:
   ```
   CHGJOBD JOBD(QMlibraryname/QMQMJOBD) JOBQ(QMQM/QMQM2)
   ```
   g. Start the subsystem by issuing the following command:
   ```
   STRSBS SBSD(QMQM/QMQM2)
   ```

   Note:

   a. You can specify the subsystem in any library. If for any reason the product is reinstalled, or the QMQM library is replaced, any changes you made are removed.
   b. All the queue manager jobs associated with the QMlibraryname now run under subsystem QMQM2.
5. The following example collects all output for a job type.

   To collect all the checkpoint process, AMQALMPX, job logs for multiple queue managers onto a single output queue, carry out the following steps:

   a. Create an output queue, for example
   ```
   CRTOUTQ OUTQ(MYLIB/CHCKPTLOGS)
   ```
   b. Create a global duplicate of the QMQM/QMQMJOBD job description, using the name of the IBM MQ process that you want to control, for example
   ```
   CRTDUPOBJ OBJ(QMQMJOBD) FROMLIB(QMQM) OBJTYPE(*JOBD) NEWOBJ(AMQALMPX)
   ```
   c. Change the output queue parameter on the job description to point to your new output queue, and change the job logging level so that all messages are written to the job log.
   ```
   CHGJOBD JOBD(QMQM/AMQALMPX) OUTQ(MYLIB/CHKPTLOGS) LOG(4 00 *SECLVL)
   ```

All IBM MQ AMQALMPX jobs, for all queue managers, use the new global AMQALMPX job description, provided that there are no local overriding job descriptions in the local queue manager library.

All job log spool files for these jobs are now written to output queue CHKPTLOGS in library MYLIB.

Note:

a. The preceding example works only if the QPJOBLOG, or any print file, has a value of *JOB for its output queue parameter. In the preceding example, the QSYS/QPDJOBLOG file needs OUTQ set to *JOB.
b. To change a system print file, use the CHGPRTF command. For example:
```
CHGPRTF PRTF(QJOBLOG) OUTQ(*JOB)
```
   The *JOB option indicates that your job descriptions must be used.
c. You can send any spool files associated with the IBM MQ jobs to a particular output queue. However, verify that the print file being used has the appropriate value for the OUTQ parameter.

Availability, backup, recovery, and restart

Use this information to understand how IBM MQ for IBM i uses the IBM i journaling support to help its backup and restore strategy.

You must be familiar with standard IBM i backup and recovery methods, and with the use of journals and their associated journal receivers on IBM i, before reading this section. For information on these topics, see Backup and recovery.

To understand the backup and recovery strategy, you first need to understand how IBM MQ for IBM i organizes its data in the IBM i file system and the integrated file system (IFS).

IBM MQ for IBM i holds its data in an individual library for each queue manager instance, and in stream files in the IFS file system.

The queue manager specific libraries contain journals, journal receivers, and objects required to control the work management of the queue manager. The IFS directories and files contain IBM MQ configuration files, the descriptions of IBM MQ objects, and the data they contain.

Every change to these objects, that is recoverable across a system failure, is recorded in a journal *before* it is applied to the appropriate object. This has the effect that such changes can be recovered by replaying the information recorded in the journal.

You can configure IBM MQ for IBM i to use multiple queue manager instances on different servers to provide increased queue manager availability and speed up recovery in the case of a server or queue manager failure.

IBM MQ for IBM i journals

Use this information to understand how IBM MQ for IBM i uses journals in its operation to control updates to local objects.

Each queue manager library contains a journal for that queue manager, and the journal has the name `QM` *GRLIB*/`AMQ` *A* `JRN`, where `QM` *GRLIB* is the name of the queue manager library, and *A* is a letter, `A` in the case of a single instance queue manager, that is unique to the queue manager instance.

`QM` *GRLIB* takes the name `QM`, followed by the name of the queue manager in a unique form. For example, a queue manager named `TEST` has a queue manager library named `QMTEST`. The queue manager library can be specified when creating a queue manager using the **CRTMQM** command.

Journals have associated journal receivers that contain the information being journaled. The receivers are objects to which information can only be appended and will fill up eventually.

Journal receivers use up valuable disk space with out-of-date information. However, you can place the information in permanent storage to minimize this problem. One journal receiver is attached to the journal at any particular time. If the journal receiver reaches its predetermined threshold size, it is detached and replaced by a new journal receiver. You can specify the threshold of journal receivers when you create a queue manager using **CRTMQM** and the **`THRESHOLD`** parameter.

The journal receivers associated with the local IBM MQ for IBM i journal exist in each queue manager library, and adopt a naming convention as follows:

`AMQ` *Arnnnnn*

where

A
: is a letter `A-Z`. It is `A` for single instance queue managers. It varies for different instances of a multi-instance queue manager.

nnnnn
: is decimal `00000 to 99999` that is incremented by 1 for the next journal in the sequence.

r
: is decimal `0 to 9`, that is incremented by 1 each time a receiver is restored.

The sequence of the journals is based on date. However, the naming of the next journal is based on the following rules:

1. `AMQArnnnnn` goes to `AMQAr(nnnnn+1)`, and `nnnnn` wraps when it reaches `99999`. For example, `AMQA099999` goes to `AMQA000000`, and `AMQA999999` goes to `AMQA900000`.
2. If a journal with a name generated by rule 1 already exists, the message `CPI70E3` is sent to the `QSYSOPR` message queue and automatic receiver switching stops.

   The currently-attached receiver continues to be used until you investigate the problem and manually attach a new receiver.
3. If no new name is available in the sequence (that is, all possible journal names are on the system) you need to do both of the following:
   a. Delete journals no longer needed (see "Journal management" on page 215).
   b. Record the journal changes into the latest journal receiver using (**`RCDMQMIMG`**) and then repeat the previous step. This allows the old journal receiver names to be reused.

The `AMQAJRN` journal uses the `MNGRCV(*SYSTEM)` option to enable the operating system to automatically change journal receivers when the threshold is reached. For more information on how the system manages receivers, see *IBM i Backup and Recovery*.

The journal receiver's default threshold value is 100,000 KB. You can set this to a larger value when you create the queue manager. The initial value of the `LogReceiverSize` attribute is written to the `LogDefaults` stanza of the `mqs.ini` file.

When a journal receiver extends beyond its specified threshold, the receiver is detached and a new journal receiver is created, inheriting attributes from the previous receiver. Changes to the `LogReceiverSize` or `LogASP` attributes after a queue manager has been created are ignored when the system automatically attaches a new journal receiver

See Changing configuration information on IBM i for further details on configuring the system.

If you need to change the size of journal receivers after the queue manager has been created, create a new journal receiver and set its owner to `QMQM` using the following commands:

```
CRTJRNRCV JRNRCV(QM GRLIB/AMQ Arnnnnn) THRESHOLD(xxxxxx) +
TEXT('MQM LOCAL JOURNAL RECEIVER')
CHGOBJOWN OBJ(QM GRLIB/AMQ Arnnnnn) OBJTYPE(*JRNRCV) NEWOWN(QMQM)
```

where

QMGRLIB
Is the name of your queue manager library

A Is the instance identifier (usually `A`).

rnnnnn
Is the next journal receiver in the naming sequence described previously

xxxxxx
Is the new receiver threshold (in KB)

Note: The maximum size of the receiver is governed by the operating system. To check this value look at the THRESHOLD keyword on the **CRTJRNRCV** command.

Now attach the new receiver to the `AMQAJRN` journal with the command:

```
CHGJRN JRN(QMGRLIB/AMQ A JRN) JRNRCV(QMGRLIB/AMQ Annnnnn)
```

See "Journal management" on page 215 for details on how to manage these journal receivers.

IBM MQ for IBM i journal usage:

Use this information to understand how IBM MQ for IBM i uses journals in its operation to control updates to local objects.

Persistent updates to message queues happen in two stages. The records representing the update are first written to the journal, then the queue file is updated.

The journal receivers can therefore become more up to date than the queue files. To ensure that restart processing begins from a consistent point, IBM MQ uses checkpoints.

A checkpoint is a point in time when the record described in the journal is the same as the record in the queue. The checkpoint itself consists of the series of journal records needed to restart the queue manager. For example, the state of all transactions (that is, units of work) active at the time of the checkpoint.

Checkpoints are generated automatically by IBM MQ. They are taken when the queue manager starts and shuts down, and after a certain number of operations are logged.

You can force a queue manager to take a checkpoint by issuing the RCDMQMIMG command against all objects on a queue manager and displaying the results, as follows:

```
RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(<Q_MGR_NAME>) DSPJRNDTA(*YES)
```

As the queues handle further messages, the checkpoint record becomes inconsistent with the current state of the queues.

When IBM MQ is restarted, it locates the latest checkpoint record in the log. This information is held in the checkpoint file that is updated at the end of every checkpoint. The checkpoint record represents the most recent point of consistency between the log and the data. The data from this checkpoint is used to rebuild the queues as they existed at the checkpoint time. When the queues are re-created, the log is then played forward to bring the queues back to the state they were in before system failure or close down.

To understand how IBM MQ uses the journal, consider the case of a local queue called `TESTQ` in the queue manager `TEST`. This is represented by the IFS file:

```
/QIBM/UserData/mqm/qmgrs/TEST/queues
```

If a specified message is put on this queue, and then retrieved from the queue, the actions that take place are shown in Figure Figure 32.

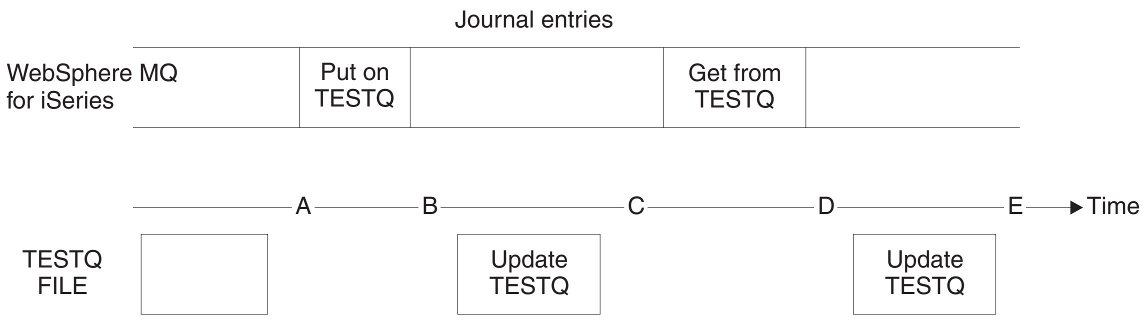


Figure 32. Sequence of events when updating MQM objects

The five points, A through E, shown in the diagram represent points in time that define the following states:

A The IFS file representation of the queue is consistent with the information contained in the journal.

B A journal entry is written to the journal defining a `Put` operation on the queue.

C The appropriate update is made to the queue.

D A journal entry is written to the journal defining a `Get` operation from the queue.

E The appropriate update is made to the queue.

The key to the recovery capabilities of IBM MQ for IBM i is that the user can save the IFS file representation of TESTQ as at time `A`, and subsequently recover the IFS file representation of TESTQ as at time `E`, by restoring the saved object and replaying the entries in the journal from time `A` onwards.

This strategy is used by IBM MQ for IBM i to recover persistent messages after system failure. IBM MQ remembers a particular entry in the journal receivers, and ensures that on startup it replays the entries in the journals from this point onwards. This startup entry is periodically recalculated so that IBM MQ only has to perform the minimum necessary replay on the next startup.

IBM MQ provides individual recovery of objects. All persistent information relating to an object is recorded in the local IBM MQ for IBM i journals. Any IBM MQ object that becomes damaged or corrupt can be completely rebuilt from the information held in the journal.

For more information on how the system manages receivers, see "Availability, backup, recovery, and restart" on page 209.

Media images:

Use this information to understand media images, and recovery from media images.

An IBM MQ object of long duration can represent a large number of journal entries, going back to the point at which it was created. To avoid this, IBM MQ for IBM i has the concept of a *media image* of an object.

This media image is a complete copy of the IBM MQ object recorded in the journal. If an image of an object is taken, the object can be rebuilt by replaying journal entries from this image onwards. The entry in the journal that represents the replay point for each IBM MQ object is referred to as its *media recovery entry*. IBM MQ keeps track of the:

- Media recovery entry for each queue manager object.
- Oldest entry from within this set (see error message AMQ7462 in "Journal management" on page 215 for details.

Images of the *CTLG object and the *MQM object are taken regularly because these objects are crucial to queue manager restart.

Images of other objects are taken when convenient. By default, images of **all** objects are taken when a queue manager is shut down using the **ENDMQM** command with parameter ENDCCTJOB(*YES). This operation can take a considerable amount of time for very large queue managers. If you need to shut down quickly, specify parameter RCDMQMIMG(*NO) with ENDCCTJOB(*YES). In such cases, you are recommended to record a complete media image in the journals after the queue manager has been restarted, using the following command:

```
RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(<Q_MGR_NAME>)
```

IBM MQ automatically records an image of an object, if it finds a convenient point at which an object can be compactly described by a small entry in the journal. However, this might never happen for some objects, for example, queues that consistently contain large numbers of messages.

Rather than allow the date of the oldest media recovery entry to continue for an unnecessarily long period, use the IBM MQ command `RCDMQMIMG`, which enables you to take an image of selected objects manually.

Recovery from media images

IBM MQ automatically recovers some objects from their media image if it is found that they are corrupt or damaged. In particular, this applies to the special *MQM and *CTLG objects as part of the normal queue manager startup. If any syncpoint transaction was incomplete at the time of the last shutdown of the queue manager, any queue affected is also recovered automatically, in order to complete the startup operation.

You must recover other objects manually, using the IBM MQ command `RCRMQMOBJ`. This command replays the entries in the journal to re-create the IBM MQ object. Should an IBM MQ object become damaged, the only valid actions are to delete it or re-create it by this method. Note, however, that nonpersistent messages cannot be recovered in this fashion.

Checkpoints:

Checkpoints are taken at various times to provide a known consistent start point for recovery.

The checkpoint process AMQALMPX is responsible for taking the checkpoint at the following points:
- Queue manager startup (STRMQM).
- Queue manager shutdown (ENDMQM).
- After a period of time has elapsed since the last checkpoint (the default period is 30 minutes) and a minimum number of log records have been written since the previous checkpoint (the default value is 100).
- After a number of log records have been written. The default value is 10 000.
- After the journal threshold size has been exceeded and a new journal receiver has been automatically created.
- When a full media image is taken with:
  ```
  RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(<Q_MGR_NAME>) DSPJRNDTA(*YES)
  ```

Backups of IBM MQ for IBM i data

Use this information to understand the two types of IBM MQ backup for each queue manager.

For each queue manager, there are two types of IBM MQ backup to consider:
- Data and journal backup.

  To ensure that both sets of data are consistent, do this only after shutting down the queue manager.
- Journal backup.

  You can do this while the queue manager is active.

For both methods, you need to find the names of the queue manager IFS directory and the queue manager library. You can find these in the IBM MQ configuration file (mqs.ini). For more information, see The QueueManager stanza.

Use the following procedures to do both types of backup:

Data and journal backup of a particular queue manager

Note: Do not use a save-while-active request when the queue manager is running. Such a request cannot complete unless all commitment definitions with pending changes are committed or rolled back. If this command is used when the queue manager is active, the channel connections might not end normally. Always use the following procedure.

1. Create an empty journal receiver, using the command:
   ```
   CHGJRN JRN(QMTEST/AMQAJRN) JRNRCV(*GEN)
   ```
2. Use the **RCDMQMIMG** command to record an MQM image for all IBM MQ objects, and then force a checkpoint using the command:
   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES) MQMNAME(TEST)
   ```
3. End channels and ensure that the queue manager is not running. If your queue manager is running, stop it with the **ENDMQM** command.
4. Backup the queue manager library by issuing the following command:
   ```
   SAVLIB LIB(QMTEST)
   ```
5. Back up the queue manager IFS directories by issuing the following command:
   ```
   SAV DEV(...) OBJ(('/QIBM/UserData/mqm/qmgrs/test'))
   ```

Journal backup of a particular queue manager

Because all relevant information is held in the journals, as long as you perform a full save at some time, partial backups can be performed by saving the journal receivers. These record all changes since the time of the full backup and are performed by issuing the following commands:

1. Create an empty journal receiver, using the command:
   ```
   CHGJRN JRN(QMTEST/AMQAJRN) JRNRCV(*GEN)
   ```
2. Use the **RCDMQMIMG** command to record an MQM image for all IBM MQ objects, and then force a checkpoint using the command:
   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES) MQMNAME(TEST)
   ```
3. Save the journal receivers using the command:
   ```
   SAVOBJ OBJ(AMQ*) LIB(QMTEST) OBJTYPE(*JRNRCV) .........
   ```

A simple backup strategy is to perform a full backup of the IBM MQ libraries every week, and perform a daily journal backup. This, of course, depends on how you have set up your backup strategy for your enterprise.

Journal management:

As part of your backup strategy, take care of your journal receivers. It is useful to remove journal receivers from the IBM MQ libraries for various reasons:
- To release space; this applies to all journal receivers
- To improve the performance when starting (STRMQM)
- To improve the performance of recreating objects (RCRMQMOBJ)

Before deleting a journal receiver, you must take care that you have a backup copy and that you no longer need the journal receiver.

Journal receivers can be removed from the queue manager library *after* they have been detached from the journals and saved, provided that they are available for restoration if needed for a recovery operation.

The concept of journal management is shown in Figure 33 on page 216.

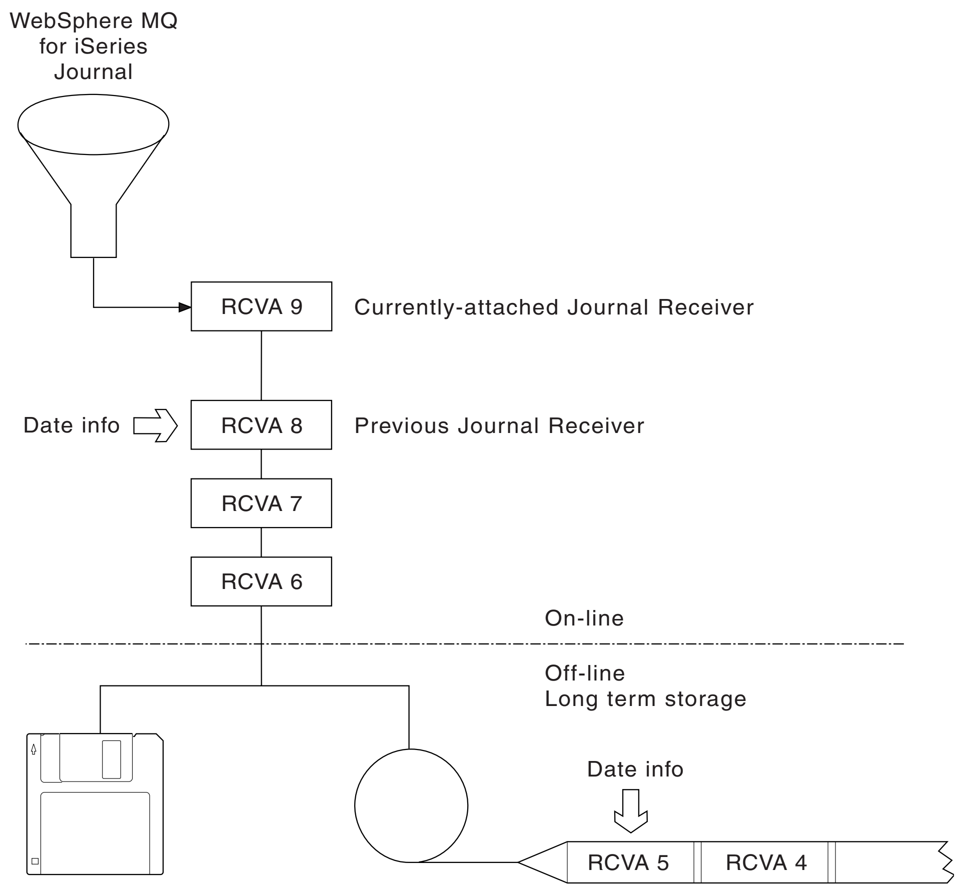

WebSphere MQ for iSeries Journal
RCVA 9
Currently-attached Journal Receiver
Date info
RCVA 8
Previous Journal Receiver
RCVA 7
RCVA 6
On-line
Off-line
Long term storage
Date info
RCVA 5
RCVA 4

Figure 33. IBM MQ for IBM i journaling

It is important to know how far back in the journals IBM MQ is likely to need to go, in order to determine when a journal receiver that has been backed up can be removed from the queue manager library, and when the backup itself can be discarded.

IBM MQ issues two messages to the queue manager message queue (QMQMMSG in the queue manager library) to help determine this time. These messages are issued when it starts, when it changes a local journal receiver, and you use RCDMQIMG to force a checkpoint. The two messages are:

AMQ7460
: Startup recovery point. This message defines the date and time of the startup entry from which IBM MQ replays the journal in the event of a startup recovery pass. If the journal receiver that contains this record is available in the IBM MQ libraries, this message also contains the name of the journal receiver containing the record.

AMQ7462
: Oldest media recovery entry. This message defines the date and time of the oldest entry to use to re-create an object from its media image.

  The journal receiver identified is the oldest one required. Any other IBM MQ journal receivers with older creation dates are no longer needed. If only stars are displayed, you need to restore backups from the date indicated to determine which is the oldest journal receiver.

When these messages are logged, IBM MQ also writes a user space object to the queue manager library that contains only one entry: the name of the oldest journal receiver that needs to be kept on the system. This user space is called AMQJRNINF, and the data is written in the format:

```
JJJJJJJJJJLLLLLLLLLLYYYYMMDDHHMMSSmmm
```

where:

`JJJJJJJJJJ`	Is the oldest receiver name that IBM MQ still needs.
`LLLLLLLLLL`	Is the journal receiver library name.
`YYYY`	Is the year of the oldest journal entry that IBM MQ needs.
`MM`	Is the month of the oldest journal entry that IBM MQ needs.
`DD`	Is the day of the oldest journal entry that IBM MQ needs.
`HH`	Is the hour of the oldest journal entry that IBM MQ needs.
`SS`	Is the seconds of the oldest journal entry that IBM MQ needs.
`mmm`	Is the milliseconds of the oldest journal entry that IBM MQ needs.

When the oldest journal receiver has been deleted from the system, this user space contains asterisks (*) for the journal receiver name.

Note: Periodically performing `RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) DSPJRNDTA(*YES)` can save startup time for IBM MQ and reduce the number of local journal receivers you need to save and restore for recovery.

IBM MQ for IBM i does not refer to the journal receivers unless it is performing a recovery pass either for startup, or for recreating an object. If it finds that a journal it requires is not present, it issues message `AMQ7432` to the queue manager message queue (QMQMMSG), reporting the time and date of the journal entry it requires to complete the recovery pass.

If this happens, restore all journal receivers that were detached after this date from the backup, to allow the recovery pass to succeed.

Keep the journal receiver that contains the startup entry, and any subsequent journal receivers, available in the queue manager library.

Keep the journal receiver containing the oldest `Media Recovery Entry`, and any subsequent journal receivers, available at all times, and either present in the queue manager library or backed-up.

When you force a checkpoint:
- If the journal receiver named in AMQ7460 is not advanced, this indicates that there is an incomplete unit of work that needs to be committed or rolled back.
- If the journal receiver named in AMQ7462 is not advanced, this indicates that there are one or more damaged objects.

Restoring a complete queue manager (data and journals):

Use this information to restore one or more queue managers from a backup or from a remote machine.

If you need to recover one or more IBM MQ queue managers from a backup, perform the following steps.

1. Quiesce the IBM MQ queue managers.
2. Locate your latest backup set, consisting of your most recent full backup and subsequently backed up journal receivers.
3. Perform a RSTLIB operation, from the full backup, to restore the IBM MQ data libraries to their state at the time of the full backup, by issuing the following commands:
   ```
   RSTLIB LIB(QMQRLIB1) .........
   RSTLIB LIB(QMQRLIB2) .........
   ```
   If a journal receiver was partially saved in one journal backup, and fully saved in a subsequent backup, restore only the fully saved one. Restore journals individually, in chronological order.
4. Perform an RST operation to restore the IBM MQ IFS directories to the IFS file system, using the following command:
   ```
   RST DEV(...) OBJ(('/QIBM/UserData/mqm/qmgrs/testqm')) ...
   ```
5. Start the message queue manager. This replays all journal records written since the full backup and restores all the IBM MQ objects to the consistent state at the time of the journal backup.

If you want to restore a complete queue manager on a different machine, use the following procedure to restore everything from the queue manager library. (We use TEST as the sample queue manager name.)

1. `CRTMQM TEST`
2. `DLTLIB LIB(QMTEST)`
3. `RSTLIB SAVLIB(QMTEST) DEV(*SAVF) SAVF(QMGRLIBSAV)`
4. Delete the following IFS files:
   ```
   /QIBM/UserData/mqm/qmgrs/TEST/QMQMCHKPT
   /QIBM/UserData/mqm/qmgrs/TEST/qmanager/QMQMOBJCAT
   /QIBM/UserData/mqm/qmgrs/TEST/qmanager/QMANAGER
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.AUTH.DATA.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CHANNEL.INITQ/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CLUSTER.COMMAND.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CLUSTER.REPOSITORY.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.CLUSTER.TRANSMIT.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.PENDING.DATA.QUEUE/q
   /QIBM/UserData/mqm/qmgrs/TEST/queues/SYSTEM.ADMIN.COMMAND.QUEUE/q
   ```
5. `STRMQM TEST`
6. `RCRMQMOBJ OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(TEST)`

Restoring journal receivers for a particular queue manager:

Use this information to understand the different ways to restore journal receivers.

The most common action is to restore a backed-up journal receiver to a queue manager library, if a receiver that has been removed is needed again for a subsequent recovery function.

This is a simple task, and requires the journal receivers to be restored using the standard IBM i RSTOBJ command:

```
RSTOBJ OBJ(QMQMDATA/AMQA000005) OBJTYPE(*JRNRCV) .........
```

A series of journal receivers might need to be restored, rather than a single receiver. For example, `AMQA000007` is the oldest receiver in the IBM MQ libraries, and both `AMQA000005` and `AMQA000006` need to be restored.

In this case, restore the receivers individually in reverse chronological order. This is not always necessary, but is good practice. In severe situations, you might need to use the IBM i command `WRKJRNA` to associate the restored journal receivers with the journal.

When restoring journals, the system automatically creates an attached journal receiver with a new name in the journal receiver sequence. However, the new name generated might be the same as a journal receiver you need to restore. Manual intervention is needed to overcome this problem; to create a new name journal receiver in sequence, and new journal before restoring the journal receiver.

For example, consider the problem with saved journal AMQAJRN and the following journal receivers:
- AMQA000000
- AMQA100000
- AMQA200000
- AMQA300000
- AMQA400000
- AMQA500000
- AMQA600000
- AMQA700000
- AMQA800000
- AMQA900000

When restoring journal AMQAJRN to a queue manager library, the system automatically creates journal receiver AMQA000000. This automatically generated receiver conflicts with one of the existing journal receivers (AMQA000000) you want to restore, which you cannot restore.

The solution is:
1. Manually create the next journal receiver (see "IBM MQ for IBM i journals" on page 210):
   ```
   CRTJRNRCV JRNRCV(QMQRLIB/AMQA900001) THRESHOLD(XXXXX)
   ```
2. Manually create the journal with the journal receiver:
   ```
   CRTJRN JRN(QMGRLIB/AMQAJRN) MNGRCV(*SYSTEM) +
   JRNRCV(QMGRLIB/AMQA9000001) MSGQ(QMGRLIB/AMQAJRNMSG)
   ```
3. Restore the local journal receivers AMQA000000 to AMQA900000.

Multi-instance queue managers

Multi-instance queue managers improve availability by automatically switching to a standby server if the active server fails. The active and standby servers are multiple instances of the same queue manager; they share the same queue manager data. If the active instance fails you need to transfer its journal to the standby that takes over so that the queue manager can rebuild its queues.

Configure the IBM i systems you are running multi-instance queue managers on so that, if the active queue manager instance fails, the journal it is using is available to the standby instance that takes over. You can design your own configuration and administration tasks to make the journal from the active instance available to the instance that takes over. If you do not want to lose messages, your design must ensure the standby journal is consistent with the active journal at the point of failure. You can adapt your design from one of the two configurations that are described with examples in subsequent topics that do maintain consistency.

1. Mirror the journal from the system that is running the active queue manager instance to the systems that are running standby instances.
2. Place the journal in an Independent Auxiliary Storage Pool (IASP) that is transferable from the system running the active instance to a standby instance.

The first solution requires no additional hardware or software as it uses basic ASPs. The second solution requires switchable IASPs which need IBM i clustering support that is available as a separately priced IBM i License Product 5761-SS1 Option 41.

Reliability and availability:

Multi-instance queue managers aim to improve the availability of applications. Technological and physical constraints mean you need different solutions to meet the demands of disaster recovery, backing up queue managers and continuous operation.

In configuring for reliability and availability you trade off a large number of factors, resulting in four distinct design points:

Disaster recovery
: Optimized for recovery after a major disaster that destroys all your local assets.

  Disaster recovery on IBM i is often based on geographic mirroring of IASP.

Backup
: Optimized for recovery after a localized failure, commonly a human error or some unforeseen technical problem.

  IBM MQ provides backup queue managers to back up queue managers periodically. You could also use asynchronous replication of queue manager journals to improve the currency of the backup.

Availability
: Optimized for restoring operations quickly giving the appearance of a nearly uninterrupted service following foreseeable technical failures such as a server or disk failure.

  Recovery is typically measured in minutes, with detection sometimes taking longer than the recovery process. A multi-instance queue manager assists you in configuring for *availability*.

Continuous operation
: Optimized for providing an uninterrupted service.

  Continuous operation solutions have to solve the detection problem, and nearly always involve submitting the same work through more than one system and either using the first result, or if correctness is a major consideration, comparing at least two outcomes.

A multi-instance queue manager assists you in configuring for *availability*. One instance of the queue manager is active at a time. Switching over to a standby instance takes from a little more than ten seconds to a fifteen minutes or more, depending on how the system is configured, loaded and tuned.

A multi-instance queue manager can give the appearance of a nearly uninterrupted service if used with reconnectable IBM MQ MQI clients, which are able to continue processing without the application program necessarily being aware of a queue manager outage; see the topic Automated client reconnection.

Components of a high availability solution:

Construct a high availability solution using multi-instance queue managers by providing robust networked storage for queue manager data, journal replication or robust IASP storage for queue manager journals, and using reconnectable clients, of applications configured as restartable queue manager services.

A multi-instance queue manager reacts to the detection of queue manager failure by resuming the startup of another queue manager instance on another server. To complete its startup, the instance needs access to the shared queue manager data in networked storage, and to its copy of the local queue manager journal.

To create a high availability solution, you need to manage the availability of the queue manager data, the currency of the local queue manager journal, and either build reconnectable client applications, or deploy your applications as queue manager services to restart automatically when the queue manager resumes. Automatic client reconnect is not supported by IBM MQ classes for Java.

Queue manager data

Place queue manager data onto networked storage that is shared, highly available and reliable, possibly by using RAID level 1 disks or greater. The file system needs to meet the requirements for a shared file system for multi-instance queue managers; for more information about the requirements for shared file systems, see Requirements for shared file systems. Network File System Version 4 (NFS4) is a protocol that meets these requirements.

Queue manager journals

You also need to configure the IBM i journals used by the queue manager instances so that the standby instance is able to restore its queue manager data to a consistent state. For uninterrupted service, this means you must restore the journals to their state when the active instance failed. Unlike backup or disaster recovery solutions, restoring journals to an earlier checkpoint is not sufficient.

You cannot physically share journals between multiple IBM i systems on networked storage. To restore queue manager journals to the consistent state at the point of failure, you either need to transfer the physical journal that was local to the active queue manager instance at the time of failure to the new instance that has been activated, or a maintain mirrors of the journal on running standby instances. The mirrored journal is a remote journal replica that has been kept exactly in sync with the local journal belonging to the failed instance.

Three configurations are starting points for designing how you manage the journals for a multi-instance queue manager,

1. Using synchronized journal replication (journal mirroring) from the active instance ASP, to the standby instances ASPs.
2. Transferring an IASP you have configured to hold the queue manager journal from the active instance to the standby instance that is taking over as the active instance.
3. Using synchronized secondary IASP mirrors.

See ASP options, for more information on putting queue manager data onto an iASP, in the IBM MQ IBM i CRTMQM command.

Also, see High availability in the IBM i Knowledge Center, and Administrator > High Availability.

Applications

To build a client to automatically reconnect to the queue manager when the standby queue manager resumes, connect your application to the queue manager using MQCONNX and specify `MQCNO_RECONNECT_Q_MGR` in the **`MQCNO`** `Options` field. See, High availability sample programs for three sample programs using reconnectable clients, and Application recovery for information about designing client applications for recovery.

Creating a network share for queue manager data using NetServer:

Create a network share on an IBM i server for storing queue manager data. Set up connections from two servers, which are going to host queue manager instances, to access the network share.

Before you begin

- You require three IBM i servers for this task. The network share is defined on one of the servers, GAMMA. The other two servers, ALPHA and BETA, are to connect to GAMMA.
- Install IBM MQ on all three servers.
- Install the System i® Navigator; see System i Navigator.

About this task

- Create the queue manager directory on GAMMA and set the correct ownership and permissions for the user profiles `QMQM` and `QMQMADM`. The directory and permission are easily created by installing IBM MQ on GAMMA.
- Use System i Navigator to create a share to the queue manager data directory on GAMMA.
- Create directories on ALPHA and BETA that point to the share.

Procedure

1. On GAMMA, create the directory to host the queue manager data with the `QMQM` user profile as the owner, and `QMQMADM` as the primary group.

   Tip:

   A quick and reliable way to create the directory with the right permissions is to install IBM MQ on GAMMA.

   Later, if you do not want to run IBM MQ on GAMMA, uninstall IBM MQ. After uninstallation, the directory `/QIBM/UserData/mqm/qmgrs` remains on GAMMA with the owner `QMQM` user profile, and `QMQMADM` the primary group.

   The task uses the `/QIBM/UserData/mqm/qmgrs` directory on GAMMA for the share.
2. Start the System i Navigator **Add connection** wizard and connect to the GAMMA system.
   a. Double-click the **System i Navigator** icon on your Windows desktop.
   b. Click **Yes** to create a connection.
   c. Follow the instructions in the **Add Connection** wizard and create a connection from the IBM i system to GAMMA.
      The connection to GAMMA is added to **My Connections**.
3. Add a new file share on GAMMA.

   a. In the System i Navigator window, click the `File Shares` folder in `My Connections/GAMMA/File Systems`.
   b. In the My Tasks window, click **Manage IBM i NetServer shares**.

      A new window, IBM i NetServer - GAMMA, opens on your desktop and shows shared objects.
   c. Right-click the `Shared Objects` folder> **File** > **New** > **File**.

      A new window, IBM i NetServer File Share - GAMMA, opens.
   d. Give the share a name, `WMQ` for example.
   e. Set the access control to `Read/Write`.
   f. Select the **Path name** by browsing to the `/QIBM/UserData/mqm/qmgrs` directory you created earlier, and click **OK**.

      The IBM i NetServer File Share - GAMMA window closes, and `WMQ` is listed in the shared objects window.
4. Right click **WMQ** in the shared objects window. Click **File** > **Permissions**.

   A window opens, Qmgrs Permissions - GAMMA, for the object `/QIBM/UserData/mqm/qmgrs`.
   a. Check the following permissions for `QMQM`, if they are not already set:
      ```
      Read
      Write
      Execute
      Management
      Existence
      Alter
      Reference
      ```
   b. Check the following permissions for `QMQMADM`, if they are not already set:
      ```
      Read
      Write
      Execute
      Reference
      ```
   c. Add other user profiles that you want to give permissions to `/QIBM/UserData/mqm/qmgrs`.

      For example, you might give the default user profile (Public) `Read` and `Execute` permissions to `/QIBM/UserData/mqm/qmgrs`.
5. Check that all the user profiles that are granted access to `/QIBM/UserData/mqm/qmgrs` on GAMMA have the same password as they do on the servers that access GAMMA.

   In particular, ensure that the `QMQM` user profiles on other servers, which are going to access the share, have the same password as the `QMQM` user profile on GAMMA.

   Tip: Click the `My Connections/GAMMA/Users and Groups` folder in the System i Navigator to set the passwords. Alternatively, use the **`CHFUSRPRF`** and **`CHGPWD`** commands.

Results

Check you can access GAMMA from other servers using the share. If you are doing the other tasks, check you can access GAMMA from ALPHA and BETA using the path `/QNTC/GAMMA/WMQ`. If the `/QNTC/GAMMA` directory does not exist on ALPHA or BETA then you must create the directory. Depending on the NetServer domain, you might have to IPL ALPHA or BETA before creating the directory.

```
CRTDIR DIR('/QNTC/GAMMA')
```

When you have checked that you have access to `/QNTC/GAMMA/WMQ` from ALPHA or BETA, issuing the command, `CRTMQM MQMNAME('QM1') MQMDIRP('/QNTC/GAMMA/WMQ')` creates `/QIBM/UserData/mqm/qmgrs/QM1` on GAMMA.

What to do next

Create a multi-instance queue manager by following the steps in either of the tasks, "Creating a multi-instance queue manager using journal mirroring and NetServer" on page 234 or "Converting a single instance queue manager to a multi-instance queue manager using NetServer and journal mirroring" on page 238.

Fail over performance:

The time it takes to detect a queue manager instance has failed, and then to resume processing on a standby can vary between tens of seconds to fifteen minutes or more depending on the configuration. Performance needs to be a major consideration in designing and testing a high availability solution.

There are advantages and disadvantages to weigh up in deciding whether to configure a multi-instance queue manager to use journal replication, or to use an IASP. Mirroring requires the queue manager to write synchronously to a remote journal. From a hardware point of view, this need not affect performance, but from a software perspective there is a greater pathlength involved in writing to a remote journal than just to a local journal, and this might be expected to reduce the performance of a running queue manager to some extent. However, when the standby queue manager takes over, the delay in synchronizing its local journal from the remote journal maintained by the active instance before it failed, is typically small in comparison to the time it takes for IBM i to detect and transfer the IASP to the server running the standby instance of the queue manager. IASP transfer times can be as much as ten to fifteen minutes rather than being completed in seconds. The IASP transfer time depends on the number of objects that need to be *varied-on* when the IASP is transferred to the standby system and the size of the access paths, or indexes, that need to be merged.

When the standby queue manager takes over, the delay in synchronizing its local journal from the remote journal maintained by the active instance before it failed, is typically small in comparison to the time it takes for IBM i to detect and transfer the independent ASP to the server running the standby instance of the queue manager. Independent ASP transfer times can be as much as ten to fifteen minutes rather than being completed in seconds. The independent ASP transfer time depends on the number of objects that need to be *varied-on* when the independent ASP is transferred to the standby system and the size of the access paths, or indices, that need to be merged.

However, transferring the journal is not the only factor influencing the time it takes for the standby instance to fully resume. You also need to consider the time it takes for the network file system to release the lock on queue manager data that signals to the standby instance to try to continue with its start-up, and also the time it takes to recover queues from the journal so that the instance is able to start processing messages again. These other sources of delay all add to the time it takes to start a standby instance. The total time to switch over consists of the following components,

Failure detection time
: The time it takes for NFS to release the lock on the queue manager data, and the standby instance to continue its startup process.

Transfer time
: In the case of an HA cluster, the time it takes IBM i to transfer the IASP from the system hosting the active instance to the standby instance, and in the case of journal replication, the time it takes to update the local journal at the standby with the data from the remote replica.

Restart time
: The time it takes for the newly active queue manager instance to rebuild its queues from the latest checkpoint in its restored journal and to resume processing messages.

Note:

If the standby instance that has taken over is configured to synchronously replicate to the previously active instance, the startup could be delayed. The new activated instance might be unable to replicate to its remote journal, if the remote journal is on the server that hosted the previously active instance, and the server has failed.

The default time to wait for a synchronous response is one minute. You can configure the maximum delay before the replication times out. Alternatively, you can configure standby instances to start using asynchronous replication to the failed active instance. Later you switch the to synchronous replication, when the failed instance is running on standby again. The same consideration applies to using synchronous independent ASP mirrors.

You can make separate baseline measurements for these components to help you assess the overall time to failover, and to factor into your decision which configuration approach to use. In making the best configuration decision you also need to consider how other applications on the same server will failover, and whether there are backup or disaster recovery processes that already use IASP.

IASP transfer times can be shortened by tuning your cluster configuration:
1. User profiles across systems in the cluster should have the same GID and UID to eliminate the need for the vary-on process to change UIDs and GIDs.
2. Minimize the number of database objects in the system and basic user disk pools, as these need to be merged to create the cross-reference table for the disk-pool group.
3. Further performance tips can be found in the IBM Redbook, *Implementing PowerHA® for IBM i, SG24-7405*.

A configuration using basic ASPs, journal mirroring, and a small configuration should switch over in the order of tens of seconds.

Overview of combining IBM i clustering capabilities with IBM MQ clustering:

Running IBM MQ on IBM i, and exploiting the IBM i clustering capabilities can provide a more comprehensive High Availability solution, than using only IBM MQ clustering.

To have this capability, you need to set up:
1. Clusters on your IBM i machine; see “IBM i clusters”
2. An independent auxiliary storage pool (IASP), into which you move the queue manager; see “Independent auxiliary storage pools (IASPs)” on page 226
3. A cluster resource group (CRG); see “Device cluster resource groups” on page 226, in which you define the:
   - Recovery domain
   - IASP
   - Exit program; see “Device CRG exit program” on page 226

IBM i clusters

An IBM i cluster is a collection of instances, that is IBM i computers or partitions, that are logically linked together.

The purpose of this grouping is to allow for each instance to be backed up, eliminating a single point of failure and increasing application and data resiliency. With a cluster created, the various cluster resource group (CRG) types can be configured to manage applications, data, and devices in the cluster.

See Creating a cluster and the Create Cluster (CRTCLU) command for further information.

Independent auxiliary storage pools (IASPs)

An IASP is a type of user ASP that serves as an extension of single-level storage. It is a piece of storage that, due to its independence from the system storage, can be easily manipulated without having to IPL the system.

An IASP can be easily switched to another operating system instance or replicated to a target IASP on another operating system instance. Two methods can be used to switch an IASP between instances:
- The first method requires all the computers in the cluster, and the switchable disk tower containing the IASP, to be connected using a High Speed Link (HSL) loop.
- The second method requires the operating system instances to be partitions on the same IBM i computer where input/output processors (IOPs) can be switched between partitions. No special hardware is needed to be able to replicate an IASP. The replication is performed using TCP/IP over the network.

See the Configure Device ASP (CFGDEVASP) command for more information.

Device cluster resource groups

There are several types of cluster resource groups (CRGs). For more information about the different types of CRGs available, see Cluster resource group.

This topic concentrates on a device CRG. A device CRG:
- Describes and manages device resources such as independent auxiliary storage pools (IASPs).
- Defines the recovery domain of the cluster nodes
- Assigns a device, and
- Assigns the exit program that will handle cluster events.

The recovery domain denotes which cluster node will be considered as the primary node. The rest of the nodes are considered to be backups. The backup nodes are also ordered in the recovery domain, specifying which node is the first backup, the second backup, and so on, depending on how many nodes there are in the recovery domain.

In the event of a primary node failure, the exit program is run on all nodes in the recovery domain. The exit program running on the first backup can then make the necessary initializations to make this node the new primary node.

See Creating device CRGs and the Create Cluster Resource Group (CRTCRG) command for more information.

Device CRG exit program

The operating system cluster resource service calls a device CRG exit program when an event occurs in one of the nodes the recovery domain defines; for example, a failover or switchover event.

A failover event occurs when the primary node of the cluster fails and the CRGs are switched with all the resources they manage, and a switchover event occurs when a specific CRG is manually switched from the primary node to the backup node.

Either way, the exit program is in charge of initializing and starting all the programs that were running on the previous primary node, which converts the first backup node into the new primary node.

For example, with IBM MQ, the exit program should be in charge of starting the IBM MQ subsystem (QMQM), and queue managers. Queue managers should be configured to automatically start listeners and services, such as trigger monitors.

Switchable IASP configuration

IBM MQ can be set up to take advantage of the clustering capabilities of IBM i. To do this:

1. Create an IBM i cluster between the data center systems
2. Move the queue manager to an IASP.

   "Moving, or removing, a queue manager to, or from, an independent auxiliary storage pool" on page 228 contains some sample code to help you carry out this operation.
3. You need to create a CRG defining the recovery domain, the IASP, and the exit program.

   "Configuring a device cluster resource group" contains some sample code to help you carry out this operation.

Related concepts:

"Independent ASPs and high availability" on page 246
Independent ASPs enable applications and data to be moved between servers. The flexibility of independent ASPs means they are the basis for some IBM i high availability solutions. In considering whether to use an ASP or independent ASP for the queue manager journal, you should consider other high availability configuration based on independent ASPs.

Configuring a device cluster resource group:

An example program to set up a device Cluster resource group (CRG).

About this task

In the following example, note that:

- [PRIMARY SITE NAME] and [BACKUP SITE NAME] could be any two distinct strings of eight characters or fewer.
- [PRIMARY IP] and [BACKUP IP] are the IPs to be used for mirroring.

Procedure

1. Identify the name of the cluster.
2. Identify the CRG exit program name and library.
3. Determine the name of the primary node and backup nodes to be defined by this CRG.
4. Identify the IASP to be managed by this CRG, and make sure it has been created under the primary node.
5. Create a device description in the backup nodes by using the command:
   ```
    CRTDEVASP DEVD([IASP NAME]) RSRCNAME([IASP NAME])
   ```
6. Add the takeover IP address to all the nodes by using the command:
   ```
   ADDTCPIFC INTNETADR(' [TAKEOVER IP]') LIND([LINE DESC])
   SUBNETMASK('[SUBNET MASK]') AUTOSTART(*NO)
   ```
7. Start the takeover IP address only in the primary node by using the command:
   ```
   STRTCPIFC INTNETADR('[TAKEOVER IP')
   ```
8. Optional: If your IASP is switchable, call this command:
   ```
   CRTCRG CLUSTER([CLUSTER NAME]) CRG( [CRG NAME]) CRGTYPE(*DEV) EXITPGM([EXIT LIB]/[EXIT NAME])
   USRPRF([EXIT PROFILE]) RCYDMN(( [PRIMARY NODE] *PRIMARY) ([BACKUP NAME] *BACKUP))
   EXITPGMFMT(EXTP0200) CFGOBJ(([IAPS NAME] *DEVD *ONLINE '[TAKEOVER IP]')
   ```
9. Optional: If your IASP is to be mirrored, call this command:

```
CRTCRG CLUSTER([CLUSTER NAME]) CRG([CRG NAME]) CRGTYPE(*DEV) EXITPGM([EXIT LIB]/[EXIT NAME])
USRPRF([EXIT PROFILE]) RCYDMN(([PRIMARY NODE] *PRIMARY *LAST [PRIMARY SITE NAME] ('[PRIMARY IP]'))
[BACKUP NAME] *BACKUP *LAST [BACKUP SITE NAME] ('[BACKUP IP]'))) EXITPGMFMT(EXTP0200)
CFGOBJ(([IAPS NAME] *DEVD *ONLINE '[TAKEOVER IP]'))
```

Moving, or removing, a queue manager to, or from, an independent auxiliary storage pool:

An example program to move a queue manager to an independent auxiliary storage pool (IASP) and commands to remove a queue manager from an IASP.

About this task

In the following example, note that:
- [MANAGER NAME] is the name of your queue manager.
- [IASP NAME] is the name of your IASP.
- [MANAGER LIBRARY] is the name of your queue manager library.
- [MANAGER DIRECTORY] is the name of your queue manager directory.

Procedure
1. Identify your primary node and your backup nodes.
2. Carry out the following procedure on your primary node:
   a. Make sure your queue manager has ended.
   b. Make sure your IASP is `vary on` by using the command
      ```
      VRYCFG CFGOBJ([IASP NAME]) CFGTYPE(*DEV) STATUS(*ON)
      ```
   c. Create the queue managers directory under the IASP. There will be a directory under root with the name of your IASP, which is:
      ```
      QSH CMD('mkdir -p /[IASP_NAME]/QIBM/UserData/mqm/qmgrs/')
      ```
   d. Move the IFS objects of your manager to the queue managers directory you have just created under the IASP using the following command:
      ```
      QSH CMD('mv /QIBM/UserData/mqm/qmgrs/[MANAGER NAME]
      /[IASP NAME]/QIBM/UserData/mqm/qmgrs')
      ```
   e. Create a temporary save file named `MGRLIB` by using the command:
      ```
      CRTSAVF QGPL/MGRLIB
      ```
   f. Save your queue manager library to the `MGRLIB` save file, by using the following command:
      ```
      SAVLIB LIB([MANGER LIBRARY]) DEV(*SAVF) SAVF(QGPL/MGRLIB)
      ```
   g. Delete the queue manager library by using the following command, and ignore all the inquiry messages:
      ```
      DLTLIB [MANAGER LIBRARY]
      ```
   h. Restore your queue manager library to the IASP by using the following command:
      ```
      RSTLIB SAVLIB([MANAGER LIBRARY]) DEV(*SAVF) SAVF(QGPL/MGRLIB)
      RSTASPDEV([IASP NAME])
      ```
   i. Delete the temporary save file by using the following command:
      ```
      DLTF FILE(QGPL/MGRLIB)
      ```
   j. Create a symbolic link to the queue manager IFS objects under the IASP, by using the following command:
      ```
      ADDLNK OBJ('/[IASP NAME]/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
      NEWLNK('/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
      ```
   k. Attach to the IASP by using the following command:
      ```
      SETASPGRP [IASP NAME]
      ```
   l. Start your queue manager by using the command:

```
STRMQM [MANAGER NAME]
```

3. Carry out the following procedure on your backup node, or nodes:
   a. Create a temporary queue manager directory by using the following command:
      ```
      QSH CMD('mkdir -p /[IASP NAME]/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
      ```
   b. Create a symbolic link to the queue manager temporary directory by using the following command:
      ```
      ADDLNK OBJ('/[IASP NAME]/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
      NEWLNK('/QIBM/UserData/mqm/qmgrs/[MANAGER NAME]')
      ```
   c. Delete the temporary directory by using the following command:
      ```
      QSH CMD('rm -r /[IASP NAME]')
      ```
   d. Add the following at the end of the file `/QIBM/UserData/mqm/mqs.ini`:
      ```
      QueueManager:
      Name=[MANAGER NAME]
      Prefix=/QIBM/UserData/mqm
      Library=[MANAGER LIBRARY]
      Directory=[MANAGER DIRECTORY]
      ```
4. To remove a queue manager from an IASP, issue the following commands:
   a. VRYCFG CFGOBJ([IASP NAME]) CFGTYPE(*DEV) STATUS(*ON)
   b. SETASPGRP [IASP NAME]
   c. ENDMQM [MANAGER NAME]
   d. DLTMQM [MANAGER NAME]

Mirrored journal configuration for ASP:

Configure a robust multi-instance queue manager using synchronous replication between mirrored journals.

A mirrored queue manager configuration uses journals that are created in basic or independent auxiliary storage pools (ASP).

On IBM i, queue manager data is written to journals and to a file system. Journals contain the master copy of queue manager data. Journals are shared between systems using either synchronous or asynchronous journal replication. A mix of local and remote journals are required to restart a queue manager instance. Queue manager restart reads journal records from the mix of local and remote journals on the server, and the queue manager data on the shared network file system. The data in the file system speeds up restarting the queue manager. Checkpoints are stored in the file system, marking points of synchronization between the file system and the journals. Journal records stored before the checkpoint are not required for typical queue manager restarts. However, the data in the file system might not be up to date, and journal records after the checkpoint are used to complete the queue manager restart. The data in the journals attached to the instance are kept up to date so that the restart can complete successfully.

But even the journal records might not be up to date, if the remote journal on the standby server was being asynchronously replicated, and the failure occurred before it was synchronized. In the event that you decide to restart a queue manager using a remote journal that is not synchronized, the standby queue manager instance might either reprocess messages that were deleted before the active instance failed, or not process messages that were received before the active instance failed.

Another, rare possibility, is that the file system contains the most recent checkpoint record, and an unsynchronized remote journal on the standby does not. In this case the queue manager does not restart automatically. You have a choice of waiting until the remote journal is synchronized, or cold starting the standby queue manager from the file system. Even though, in this case, the file system contains a more recent checkpoint of the queue manager data than the remote journal, it might not contain all the messages that were processed before the active instance failed. Some messages might be reprocessed, and some not processed, after a cold restart that is out of synchronization with the journals.

With a multi-instance queue manager, the file system is also used to control which instance of a queue manager is active, and which is the standby. The active instance acquires a lock to the queue manager data. The standby waits to acquire the lock, and when it does, it becomes the active instance. The lock is released by the active instance, if it ends normally. The lock is released by the file system if the file system detects the active instance has failed, or cannot access the file system. The file system must meet the requirements for detecting failure; see Requirements for shared file systems.

The architecture of multi-instance queue managers on IBM i provides automatic restart following server or queue manager failure. It also supports restoration of queue manager data following failure of the file system where the queue manager data is stored.

In Figure 34, if ALPHA fails, you can manually restart `QM1` on beta, using the mirrored journal. By adding the multi-instance queue manager capability to `QM1`, the standby instance of `QM1` resumes automatically on BETA if the active instance on ALPHA fails. `QM1` can also resume automatically if it is the server ALPHA that fails, not just the active instance of `QM1`. Once BETA becomes the host of the active queue manager instance, the standby instance can be started on ALPHA.

Figure 34 shows a configuration that mirrors journals between two instances of a queue manager using NetServer to store queue manager data. You might expand the pattern to include more journals, and hence more instances. Follow the journal naming rules explained in the topic, “IBM MQ for IBM i journals” on page 210. Currently the number of running instances of a queue manager is limited to two, one is active and one is in standby.

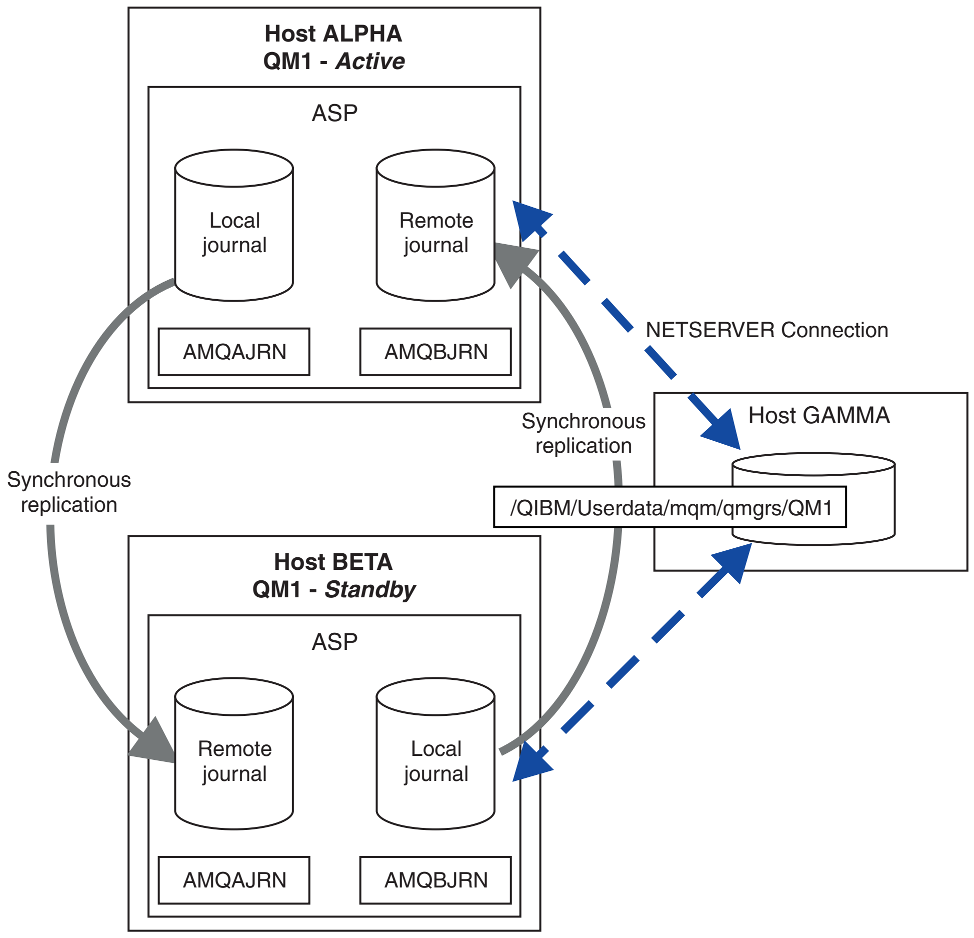


Figure 34. Mirror a queue manager journal

The local journal for QM1 on host ALPHA is called AMQAJRN (or more fully, QMQM1/AMQAJRN) and on BETA the journal is QMQM1/AMQBJRN. Each local journal replicates to remote journals on all other instances of the queue manager. If the queue manager is configured with two instances, a local journal is replicated to one remote journal.

***SYNC or *ASYNC remote journal replication**

IBM i journals are mirrored using either synchronous (*SYNC) or asynchronous (*ASYNC) journaling; see Remote journal management.

The replication mode in Figure 34 on page 230 is *SYNC, not *ASYNC. *ASYNC is faster, but if a failure occurs when the remote journal state is *ASYNCPEND, the local and remote journal are not consistent. The remote journal must catch up with the local journal. If you choose *SYNC, then the local system waits for the remote journal before returning from a call that requires a completed write. The local and remote journals generally remain consistent with one another. Only if the *SYNC operation takes longer than a designated time[1], and remote journaling is deactivated, do the journals get out of synchronization. An error is logged to the journal message queue and to QSYSOPR. The queue manager detects this message, writes an error to the queue manager error log, and deactivates remote replication of the queue manager journal. The active queue manager instance resumes without remote journaling to this journal. When the remote server is available again, you must manually reactivate synchronous remote journal replication. The journals are then resynchronized.

A problem with the *SYNC / *SYNC configuration illustrated in Figure 34 on page 230 is how the standby queue manager instance on BETA takes control. As soon as the queue manager instance on BETA writes its first persistent message, it attempts to update the remote journal on ALPHA. If the cause of control passing from ALPHA to BETA was the failure of ALPHA, and ALPHA is still down, remote journaling to ALPHA fails. BETA waits for ALPHA to respond, and then deactivates remote journaling and resumes processing messages with only local journaling. BETA has to wait a while to detect that ALPHA is down, causing a period of inactivity.

The choice between setting remote journaling to *SYNC or *ASYNC is a trade-off. Table 10 summarizes the trade-offs between using *SYNC and *ASYNC journaling between a pair of queue managers:

Table 10. Remote journaling options

Standby / Active	*SYNC	*ASYNC
***SYNC**	1. Consistent switchover and failover 2. The standby instance does not resume immediately after failover. 3. Remote journaling must be available all the time 4. Queue manager performance depends on remote journaling	1. Consistent switchover and failover 2. Remote journaling must be switched to *SYNC when standby server available 3. Remote journaling must remain available after it has been restarted 4. Queue manager performance depends on remote journaling
***ASYNC**	1. Not a sensible combination	1. Some messages might be lost or duplicated after a failover or switchover 2. Standby instance need not be available all the time for the active instance to continue without delay. 3. Performance does not depend on remote journaling

1. The designated time is 60 seconds on IBM i Version 5 and in the range 1 - 3600 seconds on IBM i 6.1 onwards.

***SYNC / *SYNC**

The active queue manager instance uses *SYNC journaling, and when the standby queue manager instance starts, it immediately tries to use *SYNC journaling.

1. The remote journal is transactionally consistent with the local journal of the active queue manager. If the queue manager is switched over to the standby instance, it can resume immediately. The standby instance normally resumes without any loss or duplication of messages. Messages are only lost or duplicated if remote journaling failed since the last checkpoint, and the previously active queue manager cannot be restarted.
2. If the queue manager fails over to the standby instance, it might not be able to start immediately. The standby queue manager instance is activated with *SYNC journaling. The cause of the failover might prevent remote journaling to the server hosting the standby instance. The queue manager waits until the problem is detected before processing any persistent messages. An error is logged to the journal message queue and to QSYSOPR. The queue manager detects this message, writes an error to the queue manager error log, and deactivates remote replication of the queue manager journal. The active queue manager instance resumes without remote journaling to this journal. When the remote server is available again, you must manually reactivate synchronous remote journal replication. The journals are then resynchronized.
3. The server to which the remote journal is replicated must always be available to maintain the remote journal. The remote journal is typically replicated to the same server that hosts the standby queue manager. The server might become unavailable. An error is logged to the journal message queue and to QSYSOPR. The queue manager detects this message, writes an error to the queue manager error log, and deactivates remote replication of the queue manager journal. The active queue manager instance resumes without remote journaling to this journal. When the remote server is available again, you must manually reactivate synchronous remote journal replication. The journals are then resynchronized.
4. Remote journaling is slower than local journaling, and substantially slower if the servers are separated by a large distance. The queue manager must wait for remote journaling, which reduces queue manager performance.

The *SYNC / *SYNC configuration between a pair of servers has the disadvantage of a delay in resuming the standby instance after failover. The *SYNC / *ASYNC configuration does not have this problem.

*SYNC / *SYNC does guarantee no message loss after switchover or failover, as long as a remote journal is available. If you want to reduce the risk of message loss after failover or switchover you have two choices. Either stop the active instance if the remote journal becomes inactive, or create remote journals on more than one server.

***SYNC / *ASYNC**

The active queue manager instance uses *SYNC journaling, and when the standby queue manager instance starts, it uses *ASYNC journaling. Shortly after the server hosting the new standby instance becomes available, the system operator must switch the remote journal on the active instance to *SYNC. When the operator switches remote journaling from *ASYNC to *SYNC the active instance pauses if the status of the remote journal is *ASYNCPEND. The active queue manager instance waits until remaining journal entries are transferred to the remote journal. When the remote journal has synchronized with the local journal, the new standby is transactionally consistent again with the new active instance. From the perspective of the management of multi-instance queue managers, in an *SYNC / *ASYNC configuration the IBM i system operator has an additional task. The operator must switch remote journaling to *SYNC in addition to restarting the failed queue manager instance.

1. The remote journal is transactionally consistent with the local journal of the active queue manager. If the active queue manager instance is switched over, or fails over to the standby instance, the standby instance can then resume immediately. The standby instance normally

resumes without any loss or duplication of messages. Messages are only lost or duplicated if remote journaling failed since the last checkpoint, and the previously active queue manager cannot be restarted.
2. The system operator must switch remote journal from `*ASYNC` to `*SYNC` shortly after the system hosting the active instance becomes available again. The operator might wait for the remote journal to catch up before switching the remote journal to `*SYNC`. Alternatively the operator might switch the remote instance to `*SYNC` immediately, and force the active instance to wait until the standby instance journal has caught up. When remote journaling is set to `*SYNC`, the standby instance is generally transactionally consistent with the active instance. Messages are only lost or duplicated if remote journaling failed since the last checkpoint, and the previously active queue manager cannot be restarted.
3. When the configuration has been restored from a switchover or failover, the server on which the remote journal is hosted must be available all the time.

Choose `*SYNC / *ASYNC` when you want the standby queue manager to resume quickly after a failover. You must restore the remote journal setting to `*SYNC` on the new active instance manually. The `*SYNC / *ASYNC` configuration matches the normal pattern of administering a pair of multi-instance queue managers. After one instance has failed, there is a time before the standby instance is restarted, during which the active instance cannot fail over.

`*ASYNC / *ASYNC`

Both the servers hosting the active and standby queue managers are configured to use `*ASYNC` remote journaling.
1. When switchover or failover take place, the queue manager continues with the journal on the new server. The journal might not be synchronized when the switchover or failover takes place. Consequently messages might be lost or duplicated.
2. The active instance runs, even if the server hosting the standby queue manager is not be available. The local journal is replicated asynchronously with the standby server when it is available.
3. The performance of the local queue manager is unaffected by remote journaling.

Choose `*ASYNC / *ASYNC` if performance is your principal requirement, and you are prepared to loose or duplicate some messages after failover or switchover.

`*ASYNC / *SYNC`

There is no reason to use this combination of options.

Queue manager activation from a remote journal

Journals are either replicated synchronously or asynchronously. The remote journal might not be active, or it might be catching up with the local journal. The remote journal might be catching up, even if it is synchronously replicated, because it might have been recently activated. The rules that the queue manager applies to the state of the remote journal it uses during start-up are as follows.
1. Standby startup fails if it must replay from the remote journal on the standby and the journal status is `*FAILED` or `*INACTPEND`.
2. When activation of the standby begins, the remote journal status on the standby must be either `*ACTIVE` or `*INACTIVE`. If the state is `*INACTIVE`, it is possible for activation to fail, if not all the journal data has been replicated.

   The failure occurs if the queue manager data on the network file system has a more recent checkpoint record than present in the remote journal. The failure is unlikely to happen, as long as the remote journal is activated well within the default 30 minute maximum interval between checkpoints. If the standby queue manager does read a more recent checkpoint record from the file system, it does not start.

   You have a choice: Wait until the local journal on the active server can be restored, or cold start the standby queue manager. If you choose to cold start, the queue manager starts with no journal data, and relies on the consistency and completeness of the queue manager data in the file system.

Note: If you cold start a queue manager, you run the risk of losing or duplicating messages after the last checkpoint. The message transactions were written to the journal, but some of the transactions might not have been written to the queue manager data in the file system. When you cold start a queue manager, a fresh journal is started, and transactions not written to the queue manager data in the file system are lost.

3. The standby queue manager activation waits for the remote journal status on the standby to change from `*ASYNCPEND` or `*SYNCPEND` to `*ASYNC` or `*SYNC`. Messages are written to the job log of the execution controller periodically.

   Note: In this case activation is waiting on the remote journal local to the standby queue manager that is being activated. The queue manager also waits for a time before continuing without a remote journal. It waits when it tries to write synchronously to its remote journal (or journals) and the journal is not available.
4. Activation stops if the journal status changes to `*FAILED` or `*INACTPEND`.

The names and states of the local and remote journals to be used in the activation are written to the queue manager error log.

Creating a multi-instance queue manager using journal mirroring and NetServer:

Create a multi-instance queue manager to run on two IBM i servers. The queue manager data is stored on a third IBM i server using NetServer. The queue manager journal is mirrored between the two servers using remote journaling. The **`ADDMQMJRN`** command is used to simplify creating the remote journals.

Before you begin

1. The task requires three IBM i servers. Install IBM MQ on two of them, ALPHA and BETA in the example. IBM MQ must be at least at version 7.0.1.1.
2. The third server is an IBM i server, connected by NetServer to ALPHA and BETA. It is used to share the queue manager data. It does not have to have an IBM MQ installation. It is useful to install IBM MQ on the server as a temporary step, to set up the queue manager directories and permissions.
3. Make sure that the `QMQM` user profile has the same password on all three servers.
4. Install IBM i NetServer; see i5/OS NetServer.

About this task

Perform the following steps to create the configuration shown in Figure 35 on page 237. The queue manager data is connected using IBM i NetServer.

- Create connections from ALPHA and BETA to the directory share on GAMMA that is to store the queue manager data. The task also sets up the necessary permissions, user profiles and passwords.
- Add Relational Database Entries (RDBE) to the IBM i systems that are going to run queue manager instances. The RDBE entries are used to connect to the IBM i systems used for remote journaling.
- Create the queue manager `QM1` on the IBM i server, ALPHA.
- Add the queue manager control information for `QM1` on the other IBM i server, BETA.
- Create remote journals on both the IBM i servers for both queue manager instances. Each queue manager writes to the local journal. The local journal is replicated to the remote journal. The command, **`ADDMQMJRN`** simplifies adding the journals and the connections.
- Start the queue manager, permitting a standby instance.

Procedure

1. Do the task, "Creating a network share for queue manager data using NetServer" on page 222.

   As a result, ALPHA and BETA have a share, `/QNTC/GAMMA/WMQ`, that points to `/QIBM/UserData/mqm/qmgrs` on GAMMA. The user profiles `QMQM` and `QMQMADM` have the necessary permissions, and `QMQM` has matching passwords on all three systems.

2. Add Relational Database Entries (RDBE) to the IBM i systems that are going to host queue manager instances.
   a. On ALPHA create the connection to BETA.
      ```
      ADDRDBDIRE RDB(BETA) RMTLOCNAME(BETA *IP) RMTAUTMTH(*USRIDPWD)
      ```
   b. On BETA create the connections to ALPHA.
      ```
      ADDRDBDIRE RDB(ALPHA) RMTLOCNAME(ALPHA *IP) RMTAUTMTH(*USRIDPWD)
      ```
3. Create the queue manager `QM1` on ALPHA, saving the queue manager data on GAMMA.
   ```
   CRTMQM MQMNAME(QM1) UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   MQMDIRP(' /QNTC/GAMMA/WMQ ')
   ```
   The path, `/QNTC/GAMMA/WMQ` , uses NetServer to create the queue manager data in `/QIBM/UserData/mqm/qmgrs`.
4. Run **`ADDMQMJRN`** on ALPHA. The command adds a remote journal on BETA for `QM1`.
   ```
   ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(BETA)
   ```
   `QM1` creates journal entries in its local journal on ALPHA when the active instance of `QM1` is on ALPHA. The local journal on ALPHA is replicated to the remote journal on BETA.
5. Use the command, **`DSPF`**, to inspect the IBM MQ configuration data created by **`CRTMQM`** for `QM1` on ALPHA.
   The information is needed in the next step.
   In this example, the following configuration is created in `/QIBM/UserData/mqm/mqs.ini` on ALPHA for `QM1`:
   ```
   Name=QM1
   Prefix=/QIBM/UserData/mqm
   Library=QMQM1
   Directory=QM1
   DataPath= /QNTC/GAMMA/WMQ /QM1
   ```
6. Create a queue manager instance of QM1 on BETA using the **`ADDMQMINF`** command. Run the following command on BETA to modify the queue manager control information in `/QIBM/UserData/mqm/mqs.ini` on BETA.
   ```
   ADDMQMINF MQMNAME(QM1)
   PREFIX('/QIBM/UserData/mqm')
   MQMDIR(QM1)
   MQMLIB(QMQM1)
   DATAPATH(' /QNTC/GAMMA/WMQ /QM1 ')
   ```
   Tip: Copy and paste the configuration information. The queue manager stanza is the same on ALPHA and BETA.
7. Run **`ADDMQMJRN`** on BETA. The command adds a local journal on BETA and a remote journal on ALPHA for `QM1`.
   ```
   ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(ALPHA)
   ```
   `QM1` creates journal entries in its local journal on BETA when the active instance of `QM1` is on BETA. The local journal on BETA is replicated to the remote journal on ALPHA.

   Note: As an alternative, you might want to set up remote journaling from BETA to ALPHA using asynchronous journaling.

   Use this command to set up asynchronous journaling from BETA to ALPHA, instead of the command in step 7.
   ```
    ADDMQMJRN MQMNAME (QM1) RMTJRNRDB (ALPHA) RMTJRNDLV (*ASYNC)
   ```

   If the server or journaling on ALPHA is the source of the failure, BETA starts without waiting for new journal entries to be replicated to ALPHA.

   Switch the replication mode to `*SYNC`, using the **`CHGMQMJRN`** command, when ALPHA is online again.

Use the information in "Mirrored journal configuration for ASP" on page 229 to decide whether to mirror the journals synchronously, asynchronously, or a mixture of both. The default is to replicate synchronously, with a 60 second wait period for a response from the remote journal.

8. Verify that the journals on ALPHA and BETA are enabled and the status of remote journal replication is `*ACTIVE`.
   a. On ALPHA:
      ```
      WRKMQMJRN MQMNAME(QM1)
      ```
   b. On BETA:
      ```
      WRKMQMJRN MQMNAME(QM1)
      ```
9. Start the queue manager instances on ALPHA and BETA.
   a. Start the first instance on ALPHA, making it the active instance. Enabling switching over to a standby instance.
      ```
      STRMQM MQMNAME(QM1) STANDBY(*YES)
      ```
   b. Start the second instance on BETA, making it the standby instance.
      ```
      STRMQM MQMNAME(QM1) STANDBY(*YES)
      ```

Results

Use **`WRKMQM`** to check queue manager status:

1. The status of the queue manager instance on ALPHA should be `*ACTIVE`.
2. The status of the queue manager instance on BETA should be `*STANDBY`.

Example

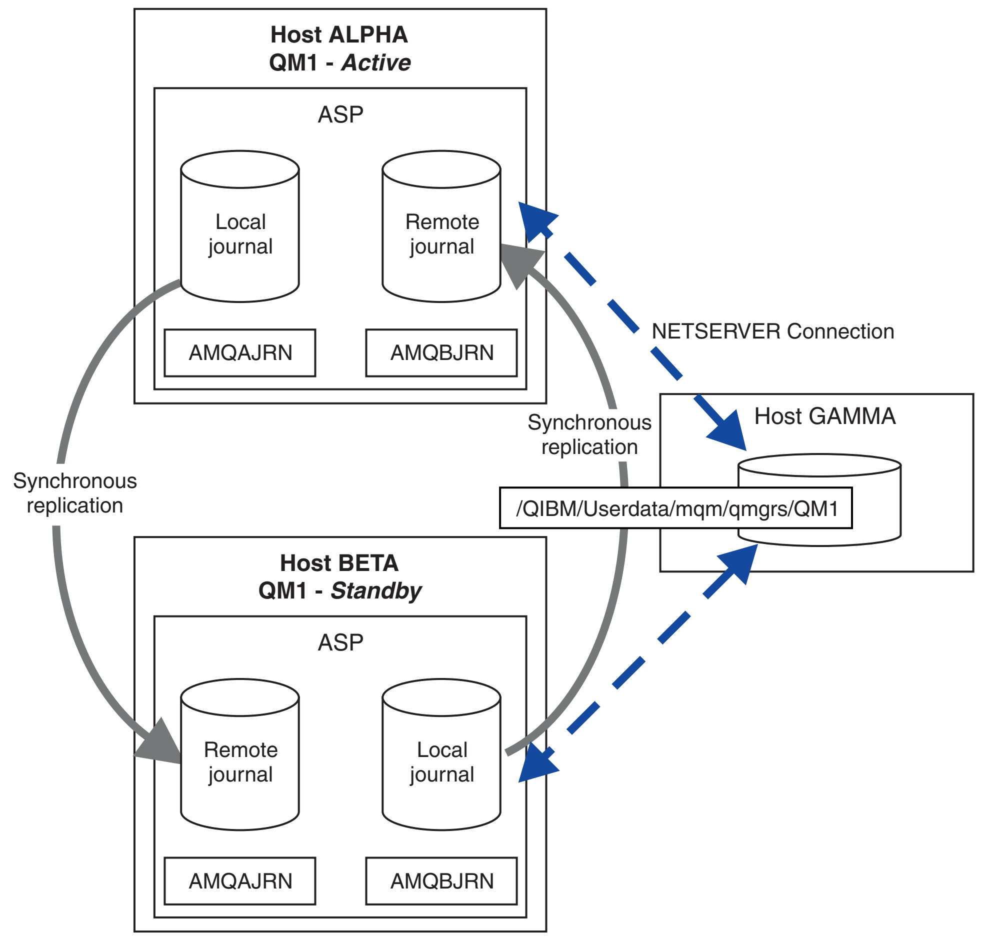


Figure 35. Mirrored journal configuration

What to do next

- Verify that the active and standby instances switch over automatically. You can run the sample high availability sample programs to test the switch over; see High availability sample programs. The sample programs are 'C' clients. You can run them from a Windows or Unix platform.
  1. Start the high availability sample programs.
  2. On ALPHA, end the queue manager requesting switch over:
     ```
     ENDMQM MQMNAME(QM1) OPTION(*IMMED) ALSWITCH(*YES)
     ```
  3. Check that the instance of QM1 on BETA is active.
  4. Restart QM1 on ALPHA
     ```
     STRMQM MQMNAME(QM1) STANDBY(*YES)
     ```
- Look at alternative high availability configurations:
  1. Use NetServer to place the queue manager data on a Windows server.
  2. Instead of using remote journaling to mirror the queue manager journal, store the journal on an independent ASP. Use IBM i clustering to transfer the independent ASP from ALPHA to BETA.

Converting a single instance queue manager to a multi-instance queue manager using NetServer and journal mirroring:

Convert a single instance queue manager to a multi-instance queue manager. Move the queue manager data to a network share connected by NetServer. Mirror the queue manager journal to a second IBM i server using remote journaling.

Before you begin

1. The task requires three IBM i servers. The existing IBM MQ installation, on the server ALPHA in the example, must be at least at Version 7.0.1.1. ALPHA is running a queue manager called `QM1` in the example.
2. Install IBM MQ on the second IBM i server, BETA in the example.
3. The third server is an IBM i server, connected by NetServer to ALPHA and BETA. It is used to share the queue manager data. It does not have to have an IBM MQ installation. It is useful to install IBM MQ on the server as a temporary step, to set up the queue manager directories and permissions.
4. Make sure that the `QMQM` user profile has the same password on all three servers.
5. Install IBM i NetServer; see i5/OS NetServer.

About this task

Perform the following steps to convert a single instance queue manager to the multi-instance queue manager shown in Figure 36 on page 241. The single instance queue manager is deleted in the task, and then recreated, storing the queue manager data on the network share connected by NetServer. This procedure is more reliable than moving the queue manager directories and files to the network share using the **`CPY`** command.

- Create connections from ALPHA and BETA to the directory share on GAMMA that is to store the queue manager data. The task also sets up the necessary permissions, user profiles and passwords.
- Add Relational Database Entries (RDBE) to the IBM i systems that are going to run queue manager instances. The RDBE entries are used to connect to the IBM i systems used for remote journaling.
- Save the queue manager logs and definitions, stop the queue manager, and delete it.
- Recreate the queue manager, storing the queue manager data on the network share on GAMMA.
- Add the second instance of the queue manager to the other server.
- Create remote journals on both the IBM i servers for both queue manager instances. Each queue manager writes to the local journal. The local journal is replicated to the remote journal. The command, **`ADDMQMJRN`** simplifies adding the journals and the connections.
- Start the queue manager, permitting a standby instance.

Note:

In step 4 on page 239 of the task, you delete the single instance queue manager, `QM1`. Deleting the queue manager deletes all the persistent messages on queues. For this reason, complete processing all the messages stored by the queue manager, before converting the queue manager. If processing all the messages is not possible, back up the queue manager library before step 4 on page 239. Restore the queue manager library after step 5 on page 239.

Note:

In step 5 on page 239 of the task, you recreate `QM1`. Although the queue manager has the same name, it has a different queue manager identifier. Queue manager clustering uses the queue manager identifier. To delete and recreate a queue manager in a cluster, you must first remove the queue manager from the cluster; see Removing a queue manager from a cluster: Alternative method or Removing a queue manager from a cluster. When you have recreated the queue manager, add it to the cluster. Although it has the same name as before, it appears to be a new queue manager to the other queue managers in the cluster.

Procedure

1. Do the task, "Creating a network share for queue manager data using NetServer" on page 222.

   As a result, ALPHA and BETA have a share, `/QNTC/GAMMA/WMQ`, that points to `/QIBM/UserData/mqm/qmgrs` on GAMMA. The user profiles `QMQM` and `QMQMADM` have the necessary permissions, and `QMQM` has matching passwords on all three systems.
2. Add Relational Database Entries (RDBE) to the IBM i systems that are going to host queue manager instances.
   a. On ALPHA create the connection to BETA.
      ```
      ADDRDBDIRE RDB(BETA) RMTLOCNAME(BETA *IP) RMTAUTMTH(*USRIDPWD)
      ```
   b. On BETA create the connections to ALPHA.
      ```
      ADDRDBDIRE RDB(ALPHA) RMTLOCNAME(ALPHA *IP) RMTAUTMTH(*USRIDPWD)
      ```
3. Create the scripts that recreate the queue manager objects.
   ```
   QSAVEQMGR LCLQMGRNAM(QM1) FILENAME('*CURLIB/QMQSC(QM1)')
   OUTPUT(*REPLACE) MAKEAUTH(*YES) AUTHFN('*CURLIB/QMAUT(QM1)')
   ```
4. Stop the queue manager and delete it.
   ```
   ENDMQM MQMNAME(QM1) OPTION(*IMMED) ENDCCTJOB(*YES) RCDMQMIMG(*YES) TIMEOUT(15)
   DLTMQM MQMNAME(QM1)
   ```
5. Create the queue manager `QM1` on ALPHA, saving the queue manager data on GAMMA.
   ```
   CRTMQM MQMNAME(QM1) UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   MQMDIRP(' /QNTC/GAMMA/WMQ ')
   ```
   The path, ` /QNTC/GAMMA/WMQ `, uses NetServer to create the queue manager data in `/QIBM/UserData/mqm/qmgrs`.
6. Recreate the queue manager objects for `QM1` from the saved definitions.
   ```
   STRMQMMQSC SRCMBR(QM1) SRCFILE(*CURLIB/QMQSC) MQMNAME(QM1)
   ```
7. Apply the authorizations from the saved information.
   a. Compile the saved authorization program.
      ```
      CRTCLPGM PGM(*CURLIB/QM1) SRCFILE(*CURLIB/QMAUT)
      SRCMBR(QM1) REPLACE(*YES)
      ```
   b. Run the program to apply the authorizations.
      ```
      CALL PGM(*CURLIB/QM1)
      ```
   c. Refresh the security information for `QM1`.
      ```
      RFRMQMAUT MQMNAME(QM1)
      ```
8. Run **`ADDMQMJRN`** on ALPHA. The command adds a remote journal on BETA for `QM1`.
   ```
   ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(BETA)
   ```
   `QM1` creates journal entries in its local journal on ALPHA when the active instance of `QM1` is on ALPHA. The local journal on ALPHA is replicated to the remote journal on BETA.
9. Use the command, **`DSPF`**, to inspect the IBM MQ configuration data created by **`CRTMQM`** for `QM1` on ALPHA.

   The information is needed in the next step.

   In this example, the following configuration is created in `/QIBM/UserData/mqm/mqs.ini` on ALPHA for `QM1`:
   ```
   Name=QM1
   Prefix=/QIBM/UserData/mqm
   Library=QMQM1
   Directory=QM1
   DataPath= /QNTC/GAMMA/WMQ /QM1
   ```

10. Create a queue manager instance of QM1 on BETA using the **ADDMQMINF** command. Run the following command on BETA to modify the queue manager control information in `/QIBM/UserData/mqm/mqs.ini` on BETA.

    ```
    ADDMQMINF MQMNAME(QM1)
    PREFIX('/QIBM/UserData/mqm')
    MQMDIR(QM1)
    MQMLIB(QMQM1)
    DATAPATH(' /QNTC/GAMMA/WMQ /QM1 ')
    ```

    Tip: Copy and paste the configuration information. The queue manager stanza is the same on ALPHA and BETA.
11. Run **ADDMQMJRN** on BETA. The command adds a local journal on BETA and a remote journal on ALPHA for `QM1`.

    ```
    ADDMQMJRN MQMNAME(QM1) RMTJRNRDB(ALPHA)
    ```

    `QM1` creates journal entries in its local journal on BETA when the active instance of `QM1` is on BETA. The local journal on BETA is replicated to the remote journal on ALPHA.

    Note: As an alternative, you might want to set up remote journaling from BETA to ALPHA using asynchronous journaling.

    Use this command to set up asynchronous journaling from BETA to ALPHA, instead of the command in step 7 on page 235.

    ```
     ADDMQMJRN MQMNAME (QM1) RMTJRNRDB (ALPHA) RMTJRNDLV (*ASYNC)
    ```

    If the server or journaling on ALPHA is the source of the failure, BETA starts without waiting for new journal entries to be replicated to ALPHA.

    Switch the replication mode to `*SYNC`, using the **CHGMQMJRN** command, when ALPHA is online again.

    Use the information in "Mirrored journal configuration for ASP" on page 229 to decide whether to mirror the journals synchronously, asynchronously, or a mixture of both. The default is to replicate synchronously, with a 60 second wait period for a response from the remote journal.
12. Verify that the journals on ALPHA and BETA are enabled and the status of remote journal replication is `*ACTIVE`.
    a. On ALPHA:

       ```
       WRKMQMJRN MQMNAME(QM1)
       ```
    b. On BETA:

       ```
       WRKMQMJRN MQMNAME(QM1)
       ```
13. Start the queue manager instances on ALPHA and BETA.
    a. Start the first instance on ALPHA, making it the active instance. Enabling switching over to a standby instance.

       ```
       STRMQM MQMNAME(QM1) STANDBY(*YES)
       ```
    b. Start the second instance on BETA, making it the standby instance.

       ```
       STRMQM MQMNAME(QM1) STANDBY(*YES)
       ```

Results

Use **WRKMQM** to check queue manager status:
1. The status of the queue manager instance on ALPHA should be `*ACTIVE`.
2. The status of the queue manager instance on BETA should be `*STANDBY`.

Example

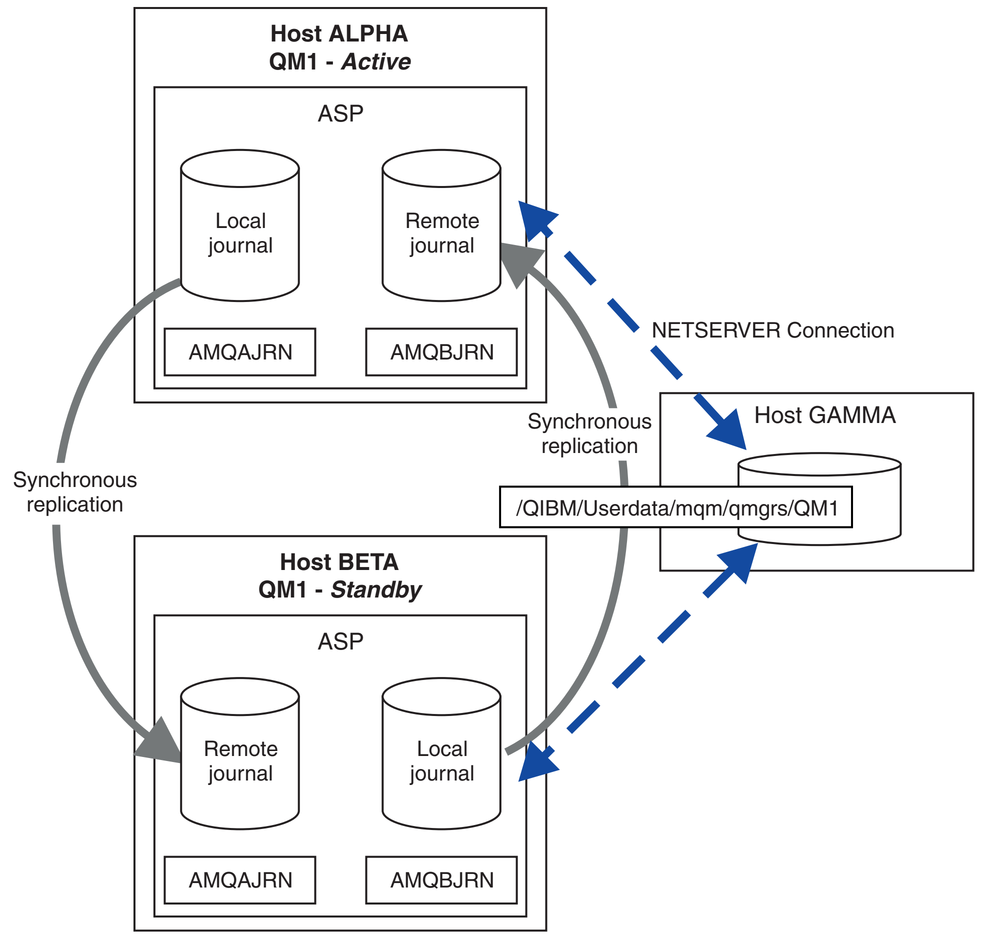


Figure 36. Mirrored journal configuration

What to do next

- Verify that the active and standby instances switch over automatically. You can run the sample high availability sample programs to test the switch over; see High availability sample programs. The sample programs are 'C' clients. You can run them from a Windows or Unix platform.
  1. Start the high availability sample programs.
  2. On ALPHA, end the queue manager requesting switch over:
     ```
     ENDMQM MQMNAME(QM1) OPTION(*IMMED) ALSWITCH(*YES)
     ```
  3. Check that the instance of `QM1` on BETA is active.
  4. Restart `QM1` on ALPHA
     ```
     STRMQM MQMNAME(QM1) STANDBY(*YES)
     ```
- Look at alternative high availability configurations:
  1. Use NetServer to place the queue manager data on a Windows server.
  2. Instead of using remote journaling to mirror the queue manager journal, store the journal on an independent ASP. Use IBM i clustering to transfer the independent ASP from ALPHA to BETA.

Switched independent ASP journal configuration:

You do not need to replicate an independent ASP journal to create a multi-instance queue manager configuration. You do need to automate a means to transfer the independent ASP from the active queue manager to the standby queue manager. There are alternative high availability solutions possible using an independent ASP, not all of which require using a multi-instance queue manager.

When using an independent ASP you do not need to mirror the queue manager journal. If you have installed cluster management, and the servers hosting the queue manager instances are in the same cluster resource group, then the queue manager journal can be transferred automatically to another server within a short distance of the active server, if the host running the active instance fails. You can also transfer the journal manually, as part of a planned switch, or you can write a command procedure to transfer the independent ASP programmatically.

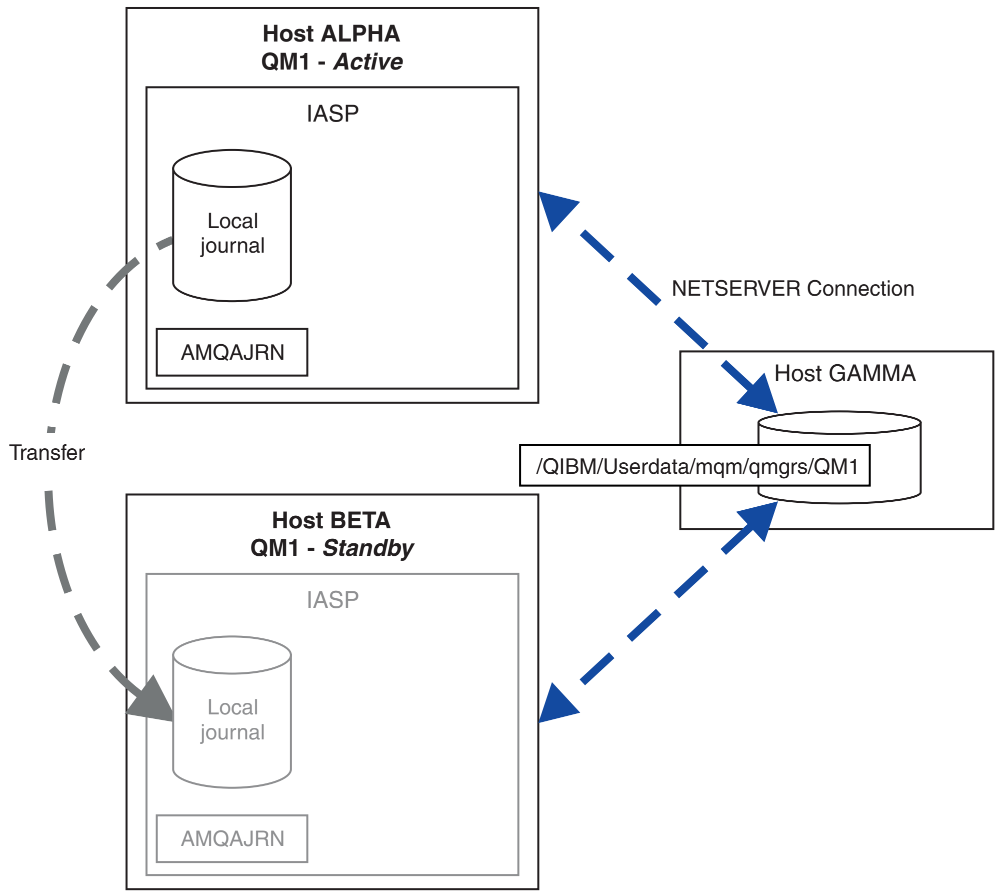


Figure 37. Transfer a queue manager journal using an independent ASP

For multi-instance queue manager operation, queue manager data must be stored on an shared file system The file system can be hosted on a variety of different platforms. You cannot store multi-instance queue manager data on an ASP or independent ASP.

The shared file system performs two roles in the configuration: The same queue manager data is shared betweem all instances of the queue manager. The file system must have a robust locking protocol that ensures only one instance of the queue manager has access to queue manager data once it has started. If the queue manager fails, or the communications to the file server breaks, then the file system must release the lock to the queue manager data held by the active instance that is no longer communicating with the file syste. The standby queue manager instance can then gain read/write access to the queue manager data. The file system protocol must conform to a set of rules to work correctly with multi-instance queue managers; see "Components of a high availability solution" on page 221.

The locking mechanism serializes the start queue manager command and controls which instance of the queue manager is active. Once a queue manager becomes active, it rebuilds its queues from the local journal that you, or the HA cluster, has transferred to the standby server. Reconnectable clients that are waiting for reconnection to the same queue manager get reconnected, and any inflight transactions are backed out. Applications that are configured to start as queue manager services are started.

You need to ensure that the local journal from the failed active queue manager instance on the independent ASP is transferred to the server that hosts the newly activated standby queue manager instance, either by configuring the cluster resource manager, or transferring the independent ASP manually. Using independent ASPs does not preclude configuring remote journals and mirroring, if you decide to use independent ASP for backup and disaster recovery, and use remote journal mirroring for multi-instance queue manager configuration.

If you have chosen to use an independent ASP, there are alternative highly available configurations you might consider. The background to these solutions are described in "Independent ASPs and high availability" on page 246.

1. Rather than use multi-instance queue managers, install and configure a single instance queue manager entirely on an independent ASP, and use IBM i high availability services to fail the queue manager over. You would probably need to augment the solution with a queue manager monitor to detect whether the queue manager has failed independently of the server. This is the basis of the solution provided in, *Supportpac MC41: Configuring IBM MQ for iSeries for High Availability*.
2. Use independent ASPs and cross site mirroring (XSM) to mirror the independent ASP rather than switching the independent ASP on the local bus. This extends the geographic range of the independent ASP solution to as far as the time taken to write log records over a long distance allows.

Creating a multi-instance queue manager using an independent ASP and NetServer:

Create a multi-instance queue manager to run on two IBM i servers. The queue manager data is stored an IBM i server using NetServer. The queue manager journal is stored on an independent ASP. Use IBM i clustering or a manual procedure to transfer the independent ASP containing the queue manager journal to the other IBM i server.

Before you begin

1. The task requires three IBM i servers. Install IBM MQ on two of them, ALPHA and BETA in the example. IBM MQ must be at least at version 7.0.1.1.
2. The third server is an IBM i server, connected by NetServer to ALPHA and BETA. It is used to share the queue manager data. It does not have to have an IBM MQ installation. It is useful to install IBM MQ on the server as a temporary step, to set up the queue manager directories and permissions.
3. Make sure that the `QMQM` user profile has the same password on all three servers.
4. Install IBM i NetServer; see i5/OS NetServer.
5. Create procedures to transfer the independent ASP from the failed queue manager to the standby that is taking over. You might find some of the techniques in *SupportPac MC41: Configuring IBM MQ for iSeries for High Availability* helpful in designing your independent ASP transfer procedures.

About this task

Perform the following steps to create the configuration shown in Figure 38 on page 245. The queue manager data is connected using IBM i NetServer.

- Create connections from ALPHA and BETA to the directory share on GAMMA that is to store the queue manager data. The task also sets up the necessary permissions, user profiles and passwords.
- Create the queue manager `QM1` on the IBM i server, ALPHA.

- Add the queue manager control information for QM1 on the other IBM i server, BETA.
- Start the queue manager, permitting a standby instance.

Procedure

1. Do the task, "Creating a network share for queue manager data using NetServer" on page 222.

   As a result, ALPHA and BETA have a share, /QNTC/GAMMA/WMQ, that points to /QIBM/UserData/mqm/qmgrs on GAMMA. The user profiles QMQM and QMQMADM have the necessary permissions, and QMQM has matching passwords on all three systems.
2. Create the queue manager QM1 on ALPHA, saving the queue manager data on GAMMA.
   ```
   CRTMQM MQMNAME(QM1) UDLMSGQ(SYSTEM.DEAD.LETTER.QUEUE)
   MQMDIRP(' /QNTC/GAMMA/WMQ ')
   ```
   The path, /QNTC/GAMMA/WMQ , uses NetServer to create the queue manager data in /QIBM/UserData/mqm/qmgrs.
3. Use the command, **DSPF**, to inspect the IBM MQ configuration data created by **CRTMQM** for QM1 on ALPHA.

   The information is needed in the next step.

   In this example, the following configuration is created in /QIBM/UserData/mqm/mqs.ini on ALPHA for QM1:
   ```
   Name=QM1
   Prefix=/QIBM/UserData/mqm
   Library=QMQM1
   Directory=QM1
   DataPath= /QNTC/GAMMA/WMQ /QM1
   ```
4. Create a queue manager instance of QM1 on BETA using the **ADDMQMINF** command. Run the following command on BETA to modify the queue manager control information in /QIBM/UserData/mqm/mqs.ini on BETA.
   ```
   ADDMQMINF MQMNAME(QM1)
   PREFIX('/QIBM/UserData/mqm')
   MQMDIR(QM1)
   MQMLIB(QMQM1)
   DATAPATH(' /QNTC/GAMMA/WMQ /QM1 ')
   ```
   Tip: Copy and paste the configuration information. The queue manager stanza is the same on ALPHA and BETA.
5. Start the queue manager instances on ALPHA and BETA.
   a. Start the first instance on ALPHA, making it the active instance. Enabling switching over to a standby instance.
   ```
   STRMQM MQMNAME(QM1) STANDBY(*YES)
   ```
   b. Start the second instance on BETA, making it the standby instance.
   ```
   STRMQM MQMNAME(QM1) STANDBY(*YES)
   ```

Results

Use **WRKMQM** to check queue manager status:

1. The status of the queue manager instance on ALPHA should be *ACTIVE.
2. The status of the queue manager instance on BETA should be *STANDBY.

Example

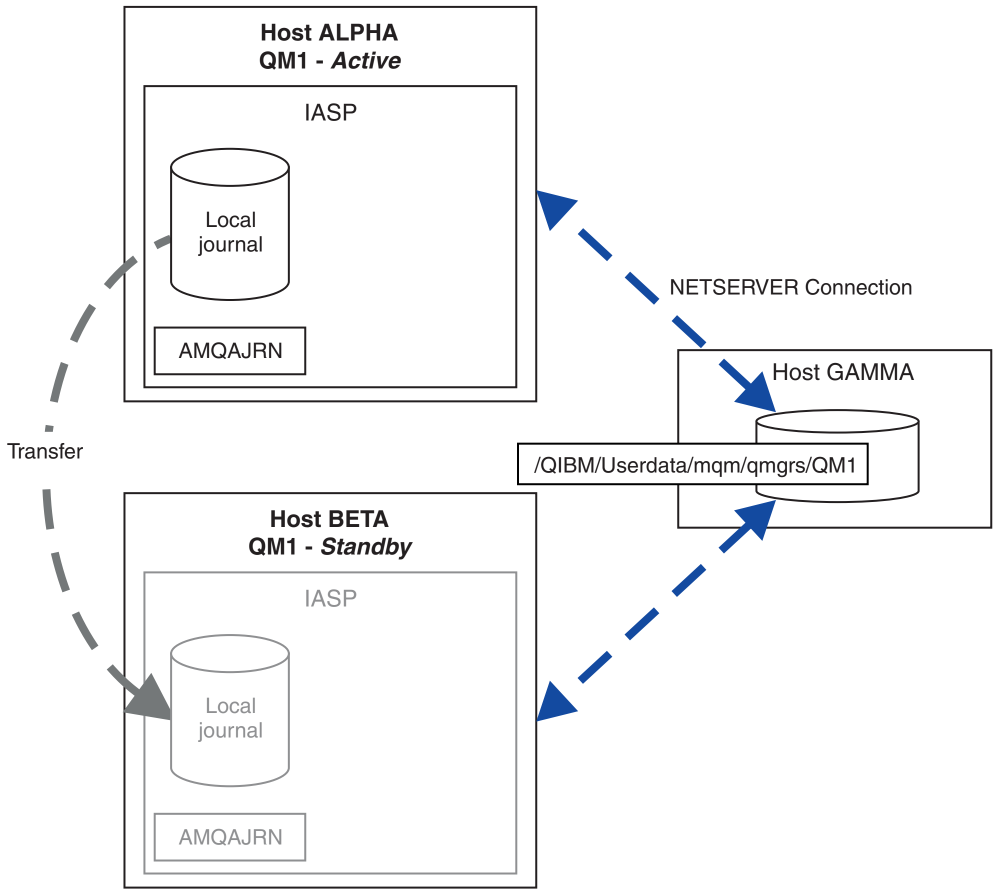


Figure 38. Transfer a queue manager journal using an independent ASP

What to do next

- Verify that the active and standby instances switch over automatically. You can run the sample high availability sample programs to test the switch over; see High availability sample programs. The sample programs are 'C' clients. You can run them from a Windows or Unix platform.
  1. Start the high availability sample programs.
  2. On ALPHA, end the queue manager requesting switch over:
     ```
     ENDMQM MQMNAME(QM1) OPTION(*IMMED) ALSWITCH(*YES)
     ```
  3. Check that the instance of QM1 on BETA is active.
  4. Restart QM1 on ALPHA
     ```
     STRMQM MQMNAME(QM1) STANDBY(*YES)
     ```
- Look at alternative high availability configurations:
  1. Use NetServer to place the queue manager data on an IBM i server.
  2. Instead of using an independent ASP to transfer the queue manager journal to the standby server, use remote journaling to mirror the journal onto the standby server.

Independent ASPs and high availability:

Independent ASPs enable applications and data to be moved between servers. The flexibility of independent ASPs means they are the basis for some IBM i high availability solutions. In considering whether to use an ASP or independent ASP for the queue manager journal, you should consider other high availability configuration based on independent ASPs.

Auxiliary storage pools (ASPs) are a building block of IBM i architecture. Disk units are grouped together to form a single ASP. By placing objects in different ASPs you can protect data in one ASP from being affected by disk failures in another ASP.

Every IBM i server has at least one *basic* ASP, known as the system ASP. It is designated as ASP1, and sometimes known as *SYSBAS. You can configure up to 31 additional basic *user* ASPs that are indistinguishable from the system ASP from the application's point of view, because they share the same namespace. By using multiple basic ASPs to distribute applications over many disks you can improve performance and reduce recovery time. Using multiple basic ASPs can also provide some degree of isolation against disk failure, but it does not improve reliability overall.

Independent ASPs are a special type of ASP. They are often called independent disk pools. Independent disk pools are key component of IBM i high availability. You can store data and applications that regard themselves as independent from the current system to which they are connected on independent disk storage units. You can configure switchable or non-switchable independent ASPs. From an availability perspective you are generally only concerned with switchable independent ASPs, which can be transferred automatically from server to server. As a result you can move the applications and data on the independent ASP from server to server.

Unlike basic user ASPs, independent ASPs do not share the same namespace as the system ASP. Applications that work with user ASPs require changes to work with an independent ASP. You need to verify your software, and third-party software you use, works in an independent ASP environment.

When the independent ASP is attached to a different server the namespace of the independent ASP has to be combined with the namespace of the system ASP. This process is called *varying-on* the independent ASP. You can vary-on an independent ASP without IPLing the server. Clustering support is required to transfer independent ASPs automatically from one server to another.

Building reliable solutions with independent ASPs

Journaling to an independent ASP, rather than journaling to an ASP and using journal replication, provides an alternative means to provide the standby queue manager with a copy of the local journal from the failed queue manager instance. To automatically transfer the independent ASP to another server you need to have installed and configured clustering support. There are a number of high-availability solutions for independent ASPs based on the cluster support, and low level disk mirroring, that you can combine with, or substitute for, using multi-instance queue managers.

The following list describes the components that are needed to build a reliable solution based on independent ASPs.

Journaling
Queue managers, and other applications, use local journals to write persistent data safely to disk to protect against loss of data in memory due to server failure. This is sometimes termed point-in-time consistency. It does not guarantee the consistency of multiple updates that take place over a period of time.

Commitment control
By using global transactions, you can coordinate updates to messages and databases so that the data written to the journal is consistent. It gives consistency over a period of time by using a two-phase commit protocol.

Switched disk

Switched disks are managed by the device cluster resource group (CRG) in an HA cluster. CRG switches independent ASPs automatically to a new server in the case of an unplanned outage. CRGs are geographically limited to the extent of the local IO bus.

By configuring your local journal on a switchable independent ASP, you can transfer the journal to a different server, and resume processing messages. No changes to persistent messages made without syncpoint control, or committed with syncpoint control, are lost, unless the independent ASP fails.

If you use both journaling and commitment control on switchable independent ASPs, you can transfer database journals and queue manager journals to a different server and resume processing transactions with no loss of consistency or committed transactions.

Cross-site mirroring (XSM)

XSM mirrors the primary independent ASP to a geographically remote secondary independent ASP across a TCP/IP network, and transfers control automatically in case of a failure. You have a choice of configuring a synchronous or asynchronous mirror. Synchronous mirroring reduces the performance of the queue manager because data is mirrored before the write operations on the production system complete, but it does guarantee the secondary independent ASP is up to date. Whereas if you use asynchronous mirroring you cannot guarantee that the secondary independent ASP is up to date. Asynchronous mirroring does maintain the consistency of the secondary independent ASP.

There are three XSM technologies.

Geographic mirroring

Geographic mirroring is an extension of clustering, enabling you to switch independent ASPs across a wide area. It has both synchronous and asynchronous modes. You can guarantee high availability only in synchronous mode, but the separation of independent ASPs might impact performance too much. You can combine geographic mirroring with switched disk to provide high availability locally and disaster recovery remotely.

Metro mirroring

Metro mirroring is a device level service that provides fast local synchronous mirroring over longer distances than the local bus. You can combine it with a multi-instance queue manager to give you high availability of the queue manager, and by having two copies of the independent ASP, high availability of the queue manager journal.

Global mirroring

Global mirroring is device level service that provides asynchronous mirroring, and is suitable for backing up and disaster recovery over longer distances, but is not an normal choice for high availability, because it only maintains point in time consistency rather than currency.

The key decision points you should consider are,

ASP or independent ASP?

You do not need to run a IBM i HA cluster to use multi-instance queue managers. You might choose independent ASPs, if you are already using independent ASPs, or you have availability requirements for other applications that require independent ASPs. It might be worth combining independent ASPs with multi-instance queue managers to replace queue manager monitoring as a means of detecting queue manager failure.

Availability?

What is the recovery time objective (RTO)? If you require the appearance of near uninterrupted behavior, then which solution has the quickest recovery time?

Journal availability?
How do you eliminate the journal as a single point of failure. You might adopt a hardware solution, using RAID 1 devices or better, or your might combine or use a software solution using replica journals or disk mirroring.

Distance?
How far apart are the active and standby queue manager instances. Can your users tolerate the performance degradation of replicating synchronously over distances greater than about 250 meters?

Skills?
There is work to be done to automate the administrative tasks involved in maintaining and exercising the solution regularly. The skills required to do the automation are different for the solutions based on ASPs and independent ASPs.

Deleting a multi-instance queue manager:

Before you delete a multi-instance queue manager, stop remote journaling, and remove queue manager instances.

Before you begin

1. In this example, two instances of the `QM1` queue manager are defined on the servers ALPHA and BETA. ALPHA is the active instance and BETA is the standby. The queue manager data associated with the queue manager `QM1` is stored on the IBM i server GAMMA, using NetServer. See "Creating a multi-instance queue manager using journal mirroring and NetServer" on page 234.
2. ALPHA and BETA must be connected so that any remote journals that are defined can be deleted by IBM MQ.
3. Verify that the `/QNTC` directory and server directory file share can be accessed, using the system commands **`EDTF`** or **`WRKLNK`**

About this task

Before you delete a multi-instance queue manager from a server using the **`DLTMQM`** command, remove any queue manager instances on other servers using the **`RMVMQMINF`** command.

When you remove a queue manager instance using the **`RMVMQMINF`** command, local and remote journals prefixed with `AMQ`, and associated with the instance, are deleted. Configuration information about the queue manager instance, local to the server, is also deleted.

Do not run the **`RMVMQMINF`** command on the server holding the remaining instance of the queue manager. Doing so prevents **`DLTMQM`** from working correctly.

Delete the queue manager using the **`DLTMQM`** command. Queue manager data is removed from the network share. Local and remote journals prefixed with `AMQ` and associated with the instance are deleted. **`DLTMQM`** also deletes configuration information about the queue manager instance, local to the server.

In the example, there are only two queue manager instances. IBM MQ supports a running multi-instance configuration that has one active queue manager instance and one standby instance. If you have created additional queue manager instances to use in running configurations, remove them, using the **`RMVMQMINF`** command, before deleting the remaining instance.

Procedure

1. Run the **`CHGMQMJRN RMTJRNSTS`** `(*INACTIVE)` command on each server to make remote journaling between the queue manager instances inactive.
   a. On ALPHA:

```
CHGMQMJRN MQMNAME('QM1')
RMTJRNRDB('BETA') RMTJRNSTS(*INACTIVE)
```

   b. On BETA:

```
CHGMQMJRN MQMNAME('QM1')
RMTJRNRDB('ALPHA') RMTJRNSTS(*INACTIVE)
```

2. Run the **ENDMQM** command on ALPHA, the active queue manager instance, to stop both instances of QM1.

   ```
   ENDMQM MQMNAME(QM1) OPTION(*IMMED) INSTANCE(*ALL) ENDCCTJOB(*YES)
   ```

3. Run the **RMVMQMINF** command on ALPHA to remove the queue manager resources for the instance from ALPHA and BETA.

   ```
   RMVMQMINF MQMNAME(QM1)
   ```

   RMVMQMINF removes the queue manager configuration information for QM1 from ALPHA. If the journal name is prefixed by AMQ, it deletes the local journal associated with QM1 from ALPHA. If the journal name is prefixed by AMQ and a remote journal has been created, it also removes the remote journal from BETA.

4. Run the **DLTMQM** command on BETA to delete QM1.

   ```
   DLTMQM MQMNAME(QM1)
   ```

   DLTMQM deletes the queue manager data from the network share on GAMMA. It removes the queue manager configuration information for QM1 from BETA. If the journal name is prefixed by AMQ, it deletes the local journal associated with QM1 from BETA. If the journal name is prefixed by AMQ and a remote journal has been created, it also removes the remote journal from ALPHA.

Results

DLTMQM and **RMVMQMINF** delete the local and remote journals created by **CRTMQM** and **ADDMQJRN**. The commands also delete the journal receivers. The journals and journal receivers must follow the naming convention of having names starting with AMQ. **DLTMQM** and **RMVMQMINF** remove the queue manager objects, queue manager data, and the queue manager configuration information from mqs.ini.

What to do next

An alternative approach is to issue the following commands after deactivating journaling in step 1 on page 248 and before ending the queue manager instances. Or, if you have not followed the naming convention, you must delete the journals and journal receivers by name.

1. On ALPHA:

   ```
   RMVMQMJRN MQMNAME('QM1') RMTJRNRDB('BETA')
   ```

2. On BETA:

   ```
   RMVMQMJRN MQMNAME('QM1') RMTJRNRDB('ALPHA')
   ```

After deleting the journals, continue with the rest of the steps.

Backing up a multi-instance queue manager:

The procedure shows you how to back up queue manager objects on the local server and the queue manager data on the network file server. Adapt the example to back up data for other queue managers.

Before you begin

In this example, the queue manager data associated with the queue manager `QM1` is stored on the IBM i server called GAMMA, using NetServer. See "Creating a multi-instance queue manager using journal mirroring and NetServer" on page 234. IBM MQ is installed on the servers, ALPHA and BETA. The queue manager, `QM1`, is configured on ALPHA and BETA.

About this task

IBM i does not support saving data from a remote directory. Save the queue manager data on a remote file system using the backup procedures local to the file system server. In this task, the network file system is on an IBM i server, GAMMA. The queue manager data is backed up in a save file on GAMMA.

If the network file system was on Windows or Linux, you might store the queue manager data in a compressed file, and then save it. If you have a back-up system, such as Tivoli Storage Manager, use it to back up the queue manager data.

Procedure

1. Create a save file on ALPHA for the queue manager library associated with `QM1`.

   Use the queue manager library name to name the save file.
   ```
   CRTSAVF FILE(QGPL/QMQM1)
   ```
2. Save the queue manager library in the save file on ALPHA.
   ```
   SAVLIB LIB(QMQM1) DEV(*SAVF) SAVF(QGPL/QMQM1)
   ```
3. Create a save file for the queue manager data directory on GAMMA.

   Use the queue manager name to name the save file.
   ```
   CRTSAVF FILE(QGPL/QMDQM1)
   ```
4. Save the copy of the queue manager data from the local directory on GAMMA.
   ```
   SAV DEV('/QSYS.LIB/QGPL.LIB/QMDQM1.FILE') OBJ('/QIBM/Userdata/mqm/qmgrs/QM1')
   ```

Commands to set up multi-instance queue managers:

IBM MQ has commands to simplify configuring journal replication, adding new queue manager instances, and configuring queue managers to use independent ASP.

The journal commands to create and manage local and remote journals are,

ADDMQMJRN
With this command you can create named local and remote journals for a queue manager instance, and configure whether replication is synchronous or asynchronous, what the synchronous timeout is, and if the remote journal is to be activated immediately.

CHGMQMJRN
The command modifies the timeout, status and delivery parameters affecting replica journals.

RMVMQMJRN
Removes named *remote* journals from a queue manager instance.

WRKMQMJRN
Lists the status of local and remote journals for a local queue manager instance.

Add and manage additional queue manager instances using the following commands, which modify the `mqs.ini` file.

ADDMQMINF
The command uses information you extracted from the `mqs.ini` file with DSPMQMINF command to add a new queue manager instance on a different IBM i server.

RMVMQMINF
Remove a queue manager instance. Use this command either to remove an instance of an existing queue manager, or to remove the configuration information for a queue manager that has been deleted from a different server.

The **CRTMQM** command has three parameters to assist configuring a multi-instance queue manager,

MQMDIRP (*DFT| ***directory-prefix*****)**
Use this parameter to select a mount point that is mapped to queue manager data on networked storage.

ASP (*SYSTEM | *ASPDEV | ***auxiliary-storage-pool-number*****)**
Specify `*SYSTEM`, or an *auxiliary-storage-pool-number* to place the queue manager journal on the system or a basic user ASP. Select the `*ASPDEV` option, and also set a device name using the **ASPDEV** parameter, to place the queue manager journal on an independent ASP.

ASPDEV (*ASP | ***device-name*****)**
Specify a *device-name* of a primary or secondary independent ASP device. Selecting `*ASP` has the same result as specifying **ASP** `(*SYSTEM)`.

Performance and disk failover considerations

Use different auxiliary storage pools to improve performance and reliability.

If you use a large number of persistent messages or large messages in your applications, the time spent writing these message to disk becomes a significant factor in the performance of the system.

Ensure that you have sufficient disk activation to cope with this possibility, or consider a separate Auxiliary Storage Pool (ASP) in which to hold your queue manager journal receivers.

You can specify which ASP your queue manager library and journals are stored on when you create your queue manager using the ASP parameter of **CRTMQM**. By default, the queue manager library and journals and IFS data are stored in the system ASP.

ASPs allow isolation of objects on one or more specific disk units. This can also reduce the loss of data because of a disk media failure. In most cases, only the data that is stored on disk units in the affected ASP is lost.

You are recommended to store the queue manager library and journal data in separate user ASPs to that of the root IFS file system to provide failover and reduce disk contention.

For more information, see Backup and recovery.

Using SAVLIB to save IBM MQ libraries

You cannot use `SAVLIB LIB(*ALLUSR)` to save the IBM MQ libraries, because these libraries have names beginning with Q.

You can use `SAVLIB LIB(QM*)` to save all the queue manager libraries, but only if you are using a save device other than *SAVF. For `DEV(*SAVF)`, you must use a SAVLIB command for each and every queue manager library on your system.

Quiescing IBM MQ for IBM i

This section explains how to quiesce (end gracefully) IBM MQ for IBM i

To quiesce IBM MQ for IBM i:

1. Sign on to a new interactive IBM MQ for IBM i session, ensuring that you are not accessing any objects.
2. Ensure that you have:
   - *ALLOBJ authority , or object management authority for the QMQM library
   - Sufficient authority to use the ENDSBS command
3. Advise all users that you are going to stop IBM MQ for IBM i.
4. How you then proceed depends on whether you want to shut down (quiesce) a single queue manager (where others might exist) (see "Shutting down a single queue manager for IBM MQ for IBM i" on page 253) or all the queue managers (see "Shutting down all queue managers for IBM MQ for IBM i" on page 254).

ENDMQM parameter ENDCCTJOB(*YES)

The ENDMQM parameter ENDCCTJOB(*YES) works differently in IBM MQ for IBM i V6.0 and later compared to previous versions.

On previous versions, when you specify ENDCCTJOB(*YES), MQ forcibly terminates your applications for you.

On IBM MQ for IBM i V6.0 or later, when you specify ENDCCTJOB(*YES), your applications are not terminated but are instead disconnected from the queue manager.

If you specify ENDCCTJOB(*YES) and you have applications that are not written to detect that a queue manager is ending, the next time a new MQI call is issued, the call will return with a MQRC_CONNECTION_BROKEN (2009) error.

As an alternative to using ENDCCTJOB(*YES), use the parameter ENDCCTJOB(*NO) and use WRKMQM option 22 (Work with jobs) to manually end any application jobs that will prevent a queue manager restart.

Shutting down a single queue manager for IBM MQ for IBM i

Use this information to understand the three types of shutdown.

In the procedures that follow, we use a sample queue manager name of `QMgr1` and a sample subsystem name of `SUBX`. Replace these names with your own values if necessary.

Planned shutdown

Planned shutdown of a queue manager on IBM i

1. Before shutdown, execute:
   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(QMgr1) DSPJRNDTA(*YES)
   ```
2. To shut down the queue manager, execute:
   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*CNTRLD)
   ```
   If QMgr1 does not end, the channel or applications are probably busy.
3. If you must shut down QMgr1 immediately, execute the following:
   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Unplanned shutdown

1. To shut down the queue manager, execute:
   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```
   If QMgr1 does not end, the channel or applications are probably busy.
2. If you need to shut down QMgr1 immediately, execute the following:
   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Shutdown under abnormal conditions

1. To shut down the queue manager, execute:
   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```
   If QMgr1 does not end, continue with step 3 providing that:
   - QMgr1 is in its own subsystem, or
   - You can end all queue managers that share the same subsystem as QMgr1. Use the unplanned shutdown procedure for all such queue managers.
2. When you have taken all the steps in the procedure for all the queue managers sharing the subsystem (`SUBX` in our examples), execute:
   ```
   ENDSBS SUBX *IMMED
   ```
   If this command fails to complete, shut down all queue managers, using the unplanned shutdown procedure, and perform an IPL on your machine.

   Warning: Do not use `ENDJOBABN` for IBM MQ jobs that fail to end as result of `ENDJOB` or `ENDSBS`, unless you are prepared to perform an IPL on your machine immediately after.
3. Start the subsystem by executing:
   ```
   STRSBS SUBX
   ```
4. Shut down the queue manager immediately, by executing:
   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(10)
   ```
5. Restart the queue manager by executing:
   ```
   STRMQM MQMNAME(QMgr1)
   ```

If this fails, and you:
- Have restarted your machine by performing an IPL, or
- Have only a single queue manager

Tidy up IBM MQ shared memory by executing:

```
ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
ENDCCTJOB(*YES) TIMEOUT(15)
```

before repeating step 5.

If the queue manager restart takes more than a few seconds, IBM MQ adds status messages intermittently to the job log detailing the startup progress.

If you still have problems restarting your queue manager, contact IBM support. Any further action you might take could damage the queue manager, leaving IBM MQ unable to recover.

Shutting down all queue managers for IBM MQ for IBM i

Use this information to understand the three types of shutdown.

The procedures are almost the same as for a single queue manager, but using `*ALL` instead of the queue manager name where possible, and otherwise using a command repeatedly using each queue manager name in turn. Throughout the procedures, we use a sample queue manager name of `QMgr1` and a sample subsystem name of `SUBX`. Replace these with your own.

Planned shutdown

1. One hour before shutdown, execute:

   ```
   RCDMQMIMG OBJ(*ALL) OBJTYPE(*ALL) MQMNAME(QMgr1) DSPJRNDTA(*YES)
   ```

   Repeat this for every queue manager that you want to shut down.
2. To shut down the queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*CNTRLD)
   ```

   Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

   If any queue manager does not end within a reasonable time (for example 10 minutes), proceed to step 3.
3. To shut down all queue managers immediately, execute the following:

   ```
   ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Unplanned shutdown

1. To shut down a queue manager, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

   Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

   If queue managers do not end, the channel or applications are probably busy.
2. If you need to shut down the queue managers immediately, execute the following:

   ```
   ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

Shutdown under abnormal conditions

1. To shut down the queue managers, execute:

   ```
   ENDMQM MQMNAME(QMgr1) OPTION(*IMMED)
   ```

Repeat this for every queue manager that you want to shut down; separate commands can run in parallel.

2. End the subsystems (`SUBX` in our examples), by executing:

   ```
   ENDSBS SUBX *IMMED
   ```

   Repeat this for every subsystem that you want to shut down; separate commands can run in parallel.

   If this command fails to complete, perform an IPL on your system.

   Warning: Do not use `ENDJOBABN` for jobs that fail to end as result of `ENDJOB` or `ENDSBS`, unless you are prepared to perform an IPL on your system immediately after.

3. Start the subsystems by executing:

   ```
   STRSBS SUBX
   ```

   Repeat this for every subsystem that you want to start.

4. Shut the queue managers down immediately, by executing:

   ```
   ENDMQM MQMNAME(*ALL) OPTION(*IMMED)
   ENDCCTJOB(*YES) TIMEOUT(15)
   ```

5. Restart the queue managers by executing:

   ```
   STRMQM MQMNAME(QMgr1)
   ```

   Repeat this for every queue manager that you want to start.

   If any queue manager restart takes more than a few seconds IBM MQ will show status messages intermittently detailing the startup progress.

   If you still have problems restarting any queue manager, contact IBM support. Any further action you might take could damage the queue managers, leaving MQSeries or IBM MQ unable to recover.

Administering IBM MQ for z/OS

z/OS

Administering queue managers and associated resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your queue managers and associated resources.

IBM MQ for z/OS can be controlled and managed by a set of utilities and programs provided with the product. You can use the IBM MQ Script (MQSC) commands or Programmable Command Formats (PCFs) to administer IBM MQ for z/OS. For information about using commands for IBM MQ for z/OS, see "Issuing commands to IBM MQ for z/OS" on page 256.

IBM MQ for z/OS also provides a set of utility programs to help you with system administration. For information about the different utility programs and how to use them, see "The IBM MQ for z/OS utilities" on page 264.

For details of how to administer IBM MQ for z/OS and the different administrative tasks you might have to undertake, see the following links:

Related concepts:
"Administering local IBM MQ objects" on page 69
This section tells you how to administer local IBM MQ objects to support application programs that use the Message Queue Interface (MQI). In this context, local administration means creating, displaying, changing, copying, and deleting IBM MQ objects.
"Administering remote IBM MQ objects" on page 129

"Administering IBM MQ" on page 1
Administering queue managers and associated resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your queue managers and associated resources.
Related information:
IBM MQ for z/OS concepts
Planning
Planning your IBM MQ environment on z/OS
Configuring
Configuring z/OS
Programmable command formats reference
MQSC reference
Using the IBM MQ for z/OS utilities

Issuing commands to IBM MQ for z/OS

You can use IBM MQ script commands (MQSC) in batch or interactive mode to control a queue manager.

IBM MQ for z/OS supports MQSC commands, which can be issued from the following sources:
- The z/OS console or equivalent (such as SDSF/TSO).
- The initialization input data sets.
- The supplied batch utility, CSQUTIL, processing a list of commands in a sequential data set.
- A suitably authorized application, by sending a command as a message to the command input queue. The application can be any of the following:
  - A batch region program
  - A CICS application
  - An IMS™ application
  - A TSO application
  - An application program or utility on another IBM MQ system

Table 12 on page 260 summarizes the MQSC commands and the sources from which they can be issued.

Much of the functionality of these commands is available in a convenient way from the IBM MQ for z/OS operations and controls panels.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of the IBM MQ subsystem.

IBM MQ for z/OS also supports Programmable Command Format (PCF) commands. These simplify the creation of applications for the administration of IBM MQ. MQSC commands are in human-readable text form, whereas PCF enables applications to create requests and read the replies without having to parse text strings. Like MQSC commands, applications issue PCF commands by sending them as messages to the command input queue. For more information about using PCF commands and for details of the commands, see the Programmable command formats reference documentation.

Private and global definitions

When you define an object on IBM MQ for z/OS, you can choose whether you want to share that definition with other queue managers (a *global* definition), or whether the object definition is to be used by one queue manager only (a *private* definition). This is called the object *disposition*.

Global definition

If your queue manager belongs to a queue-sharing group, you can choose to share any object definitions you make with the other members of the group. This means that you have to define an object once only, reducing the total number of definitions required for the whole system.

Global object definitions are held in a *shared repository* (a Db2 shared database), and are available to all the queue managers in the queue-sharing group. These objects have a disposition of GROUP.

Private definition

If you want to create an object definition that is required by one queue manager only, or if your queue manager is not a member of a queue-sharing group, you can create object definitions that are not shared with other members of a queue-sharing group.

Private object definitions are held on page set zero of the defining queue manager. These objects have a disposition of QMGR.

You can create private definitions for all types of IBM MQ objects except CF structures (that is, channels, namelists, process definitions, queues, queue managers, storage class definitions, and authentication information objects), and global definitions for all types of objects except queue managers.

IBM MQ automatically copies the definition of a group object to page set zero of each queue manager that uses it. You can alter the copy of the definition temporarily if you want, and IBM MQ allows you to refresh the page set copies from the repository copy if required.

IBM MQ always tries to refresh the page set copies from the repository copy on start up (for channel commands, this is done when the channel initiator restarts), or if the group object is changed.

Note: The copy of the definition is refreshed from the definition of the group, only if the definition of the group has changed after you created the copy of the definition.

This ensures that the page set copies reflect the version on the repository, including any changes that were made when the queue manager was inactive. The copies are refreshed by generating DEFINE REPLACE commands, therefore there are circumstances under which the refresh is not performed, for example:

- If a copy of the queue is open, a refresh that changes the usage of the queue fails.
- If a copy of a queue has messages on it, a refresh that deletes that queue fails.
- If a copy of a queue would require ALTER with FORCE to change it.

In these circumstances, the refresh is not performed on that copy, but is performed on the copies on all other queue managers.

If the queue manager is shut down and then restarted stand-alone, any local copies of objects are deleted, unless for example, the queue has associated messages.

There is a third object disposition that applies to local queues only. This allows you to create shared queues. The definition for a shared queue is held on the shared repository and is available to all the queue managers in the queue-sharing group. In addition, the messages on a shared queue are also available to all the queue managers in the queue sharing group. This is described in Shared queues and queue-sharing groups. Shared queues have an object disposition of SHARED.

The following table summarizes the effect of the object disposition options for queue managers started stand-alone, and as a member of a queue-sharing group.

Disposition	Stand-alone queue manager	Member of a queue-sharing group
QMGR	Object definition held on page set zero.	Object definition held on page set zero.
GROUP	Not allowed.	Object definition held in the shared repository. Local copy held on page set zero of each queue manager in the group.
SHARED	Not allowed.	Queue definition held in the shared repository. Messages available to any queue manager in the group.

Manipulating global definitions

If you want to change the definition of an object that is held in the shared repository, you need to specify whether you want to change the version on the repository, or the local copy on page set zero. Use the object disposition as part of the command to do this.

Directing commands to different queue managers

You can use the *command scope* to control on which queue manager the command runs.

You can choose to execute a command on the queue manager where it is entered, or on a different queue manager in the queue-sharing group. You can also choose to issue a particular command in parallel on all the queue managers in a queue-sharing group. This is possible for both MQSC commands and PCF commands.

This is determined by the *command scope*. The command scope is used with the object disposition to determine which version of an object you want to work with.

For example, you might want to alter some of the attributes of an object, the definition of which is held in the shared repository.

- You might want to change the version on one queue manager only, and not make any changes to the version on the repository or those in use by other queue managers.
- You might want to change the version in the shared repository for future users, but leave existing copies unchanged.
- You might want to change the version in the shared repository, but also want your changes to be reflected immediately on all the queue managers in the queue-sharing group that hold a copy of the object on their page set zero.

Use the command scope to specify whether the command is executed on this queue manager, another queue manager, or all queue managers. Use the object disposition to specify whether the object you are manipulating is in the shared repository (a group object), or is a local copy on page set zero (a queue manager object).

You do not have to specify the command scope and object disposition to work with a shared queue because every queue manager in the queue-sharing group handles the shared queue as a single queue.

Command summary

Use this topic as a reference of the main MQSC and PCF commands.

Table 11 summarizes the MQSC and PCF commands that are available on IBM MQ for z/OS to alter, define, delete and display IBM MQ objects.

Table 11. Summary of the main MQSC and PCF commands by object type

MQSC command	**ALTER**	**DEFINE**	**DISPLAY**	**DELETE**
PCF command	**Change**	**Create/Copy**	**Inquire**	**Delete**
AUTHINFO	X	X	X	X
CFSTATUS			X	
CFSTRUCT	X	X	X	X
CHANNEL	X	X	X	X
CHSTATUS			X	
NAMELIST	X	X	X	X
PROCESS	X	X	X	X
QALIAS	M	M	M	M
QCLUSTER			M	
QLOCAL	M	M	M	M
QMGR	X		X	
QMODEL	M	M	M	M
QREMOTE	M	M	M	M
QUEUE	P	P	X	P
QSTATUS			X	
STGCLASS	X	X	X	X

Key to table symbols:

- M = MQSC only
- P = PCF only
- X = both

There are many other MQSC and PCF commands which allow you to manage other IBM MQ resources, and carry out other actions in addition to those summarized in Table 11.

Table 12 on page 260 shows every MQSC command, and where each command can be issued from:

- CSQINP1 initialization input data set
- CSQINP2 initialization input data set
- z/OS console (or equivalent)
- SYSTEM.COMMAND.INPUT queue and command server (from applications, CSQUTIL, or the CSQINPX initialization input data set)

Table 12. Sources from which to run MQSC commands

Command	CSQINP1	CSQINP2	z/OS console	Command input queue and server
ALTER AUTHINFO		X	X	X
ALTER BUFFPOOL		X	X	X
ALTER CFSTRUCT		X	X	X
ALTER CHANNEL		X	X	X
ALTER NAMELIST		X	X	X
ALTER PSID			X	X
ALTER PROCESS		X	X	X
ALTER QALIAS		X	X	X
ALTER QLOCAL		X	X	X
ALTER QMGR		X	X	X
ALTER QMODEL		X	X	X
ALTER QREMOTE		X	X	X
ALTER SECURITY	X	X	X	X
ALTER STGCLASS		X	X	X
ALTER SUB		X	X	X
ALTER TOPIC		X	X	X
ALTER TRACE	X	X	X	X
ARCHIVE LOG	X	X	X	X
BACKUP CFSTRUCT			X	X
CLEAR QLOCAL		X	X	X
DEFINE AUTHINFO		X	X	X
DEFINE BUFFPOOL	X	X		
DEFINE CFSTRUCT		X	X	X
DEFINE CHANNEL		X	X	X
DEFINE LOG			X	X
DEFINE NAMELIST		X	X	X
DEFINE PROCESS		X	X	X
DEFINE PSID	X		X	X
DEFINE QALIAS		X	X	X
DEFINE QLOCAL		X	X	X
DEFINE QMODEL		X	X	X
DEFINE QREMOTE		X	X	X
DEFINE STGCLASS		X	X	X
DEFINE SUB			X	X
DEFINE TOPIC		X	X	X
DELETE AUTHINFO		X	X	X
DELETE BUFFPOOL			X	X
DELETE CFSTRUCT		X	X	X
DELETE CHANNEL			X	X

Table 12. Sources from which to run MQSC commands (continued)

Command	**CSQINP1**	**CSQINP2**	**z/OS console**	**Command input queue and server**
DELETE NAMELIST		X	X	X
DELETE PROCESS		X	X	X
DELETE PSID			X	X
DELETE QALIAS		X	X	X
DELETE QLOCAL		X	X	X
DELETE QMODEL		X	X	X
DELETE QREMOTE		X	X	X
DELETE STGCLASS		X	X	X
DELETE SUB		X	X	X
DELETE TOPIC		X	X	X
DISPLAY ARCHIVE	X	X	X	X
DISPLAY AUTHINFO		X	X	X
DISPLAY CFSTATUS			X	X
DISPLAY CFSTRUCT		X	X	X
DISPLAY CHANNEL		X	X	X
DISPLAY CHSTATUS			X	X
DISPLAY CLUSQMGR			X	X
DISPLAY CMDSERV	X	X	X	X
DISPLAY CONN		X	X	X
DISPLAY CHINIT		X	X	X
DISPLAY GROUP		X	X	X
DISPLAY LOG	X	X	X	X
DISPLAY NAMELIST		X	X	X
DISPLAY PROCESS		X	X	X
DISPLAY QALIAS		X	X	X
DISPLAY QCLUSTER		X	X	X
DISPLAY QLOCAL		X	X	X
DISPLAY QMGR		X	X	X
DISPLAY QMODEL		X	X	X
DISPLAY QREMOTE		X	X	X
DISPLAY QSTATUS		X	X	X
DISPLAY QUEUE		X	X	X
DISPLAY SECURITY			X	X
DISPLAY STGCLASS		X	X	X
DISPLAY SUB		X	X	X
DISPLAY TOPIC		X	X	X
DISPLAY SYSTEM	X	X	X	X
DISPLAY THREAD		X	X	X
DISPLAY TRACE	X	X	X	X

Table 12. Sources from which to run MQSC commands (continued)

Command	CSQINP1	CSQINP2	z/OS console	Command input queue and server
DISPLAY USAGE		X	X	X
MOVE QLOCAL		X	X	X
PING CHANNEL			X	X
RECOVER BSDS	X	X	X	X
RECOVER CFSTRUCT			X	X
REFRESH CLUSTER		X	X	X
REFRESH QMGR		X	X	X
REFRESH SECURITY		X	X	X
RESET CHANNEL			X	X
RESET CLUSTER		X	X	X
RESET QSTATS		X	X	X
RESET TPIPE			X	X
RESOLVE CHANNEL			X	X
RESOLVE INDOUBT		X	X	X
RESUME QMGR			X	X
RVERIFY SECURITY		X	X	X
SET ARCHIVE	X	X	X	X
SET LOG	X	X	X	X
SET SYSTEM	X	X	X	X
START CHANNEL			X	X
START CHINIT		X	X	X
START CMDSERV	X	X	X	
START LISTENER			X	X
START QMGR			X	
START TRACE	X	X	X	X
STOP CHANNEL			X	X
STOP CHINIT			X	X
STOP CMDSERV	X	X	X	
STOP LISTENER			X	X
STOP QMGR			X	X
STOP TRACE	X	X	X	X
SUSPEND QMGR			X	X

In MQSC commands, each command description identifies the sources from which that command can be run.

Initialization commands

Initialization commands can be used to control the queue manager startup.

Commands in the initialization input data sets are processed when IBM MQ is initialized on queue manager startup. Three types of command can be issued from the initialization input data sets:

- Commands to define IBM MQ entities that cannot be defined elsewhere, for example DEFINE BUFFPOOL.

  These commands must reside in the data set identified by the DD name CSQINP1. They are processed before the restart phase of initialization. They cannot be issued through the console, operations and control panels, or an application program. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT1 statement of the started task procedure.
- Commands to define IBM MQ objects that are recoverable after restart. These definitions must be specified in the data set identified by the DD name CSQINP2. They are stored in page set zero. CSQINP2 is processed after the restart phase of initialization. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT2 statement of the started task procedure.
- Commands to manipulate IBM MQ objects. These commands must also be specified in the data set identified by the DD name CSQINP2. For example, the IBM MQ-supplied sample contains an ALTER QMGR command to specify a dead-letter queue for the subsystem. The response to these commands is written to the CSQOUT2 output data set.

Note: If IBM MQ objects are defined in CSQINP2, IBM MQ attempts to redefine them each time the queue manager is started. If the objects already exist, the attempt to define them fails. If you need to define your objects in CSQINP2, you can avoid this problem by using the REPLACE parameter of the DEFINE commands, however, this overrides any changes that were made during the previous run of the queue manager.

Sample initialization data set members are supplied with IBM MQ for z/OS. They are described in Sample definitions supplied with IBM MQ.

Initialization commands for distributed queuing

You can also use the CSQINP2 initialization data set for the START CHINIT command. If you need a series of other commands to define your distributed queuing environment (for example, starting listeners), IBM MQ provides a third initialization input data set, called CSQINPX, that is processed as part of the channel initiator started task procedure.

The MQSC commands contained in the data set are executed at the end of channel initiator initialization, and output is written to the data set specified by the CSQOUTX DD statement. You might use the CSQINPX initialization data set to start listeners for example.

A sample channel initiator initialization data set member is supplied with IBM MQ for z/OS. It is described in Sample definitions supplied with IBM MQ.

Initialization commands for publish/Subscribe

If you need a series of commands to define your publish/subscribe environment (for example, when defining subscriptions), IBM MQ provides a fourth initialization input data set, called CSQINPT.

The MQSC commands contained in the data set are executed at the end of publish/subscribe initialization, and output is written to the data set specified by the CSQOUTT DD statement. You might use the CSQINPT initialization data set to define subscriptions for example.

A sample publish/subscribe initialization data set member is supplied with IBM MQ for z/OS. It is described in Sample definitions supplied with IBM MQ.

The IBM MQ for z/OS utilities

IBM MQ for z/OS provides a set of utility programs that you can use to help with system administration.

IBM MQ for z/OS supplies a set of utility programs to help you perform various administrative tasks, including the following:
- Manage message security policies.
- Perform backup, restoration, and reorganization tasks.
- Issue commands and process object definitions.
- Generate data-conversion exits.
- Modify the bootstrap data set.
- List information about the logs.
- Print the logs.
- Set up Db2 tables and other Db2 utilities.
- Process messages on the dead-letter queue.

The message security policy utility

The message security policy utility (CSQ0UTIL) runs as a stand-alone utility to manage message security policies. See The message security policy utility (CSQ0UTIL) for more information.

The CSQUTIL utility

This is a utility program provided to help you with backup, restore and reorganize tasks. See The CSQUTIL utility for more information.

The data conversion exit utility

The IBM MQ for z/OS data conversion exit utility (`CSQUCVX`) runs as a stand-alone utility to create data conversion exit routines.

The change log inventory utility

The IBM MQ for z/OS change log inventory utility program (`CSQJU003`) runs as a stand-alone utility to change the bootstrap data set (BSDS). You can use the utility to perform the following functions:
- Add or delete active or archive log data sets.
- Supply passwords for archive logs.

The print log map utility

The IBM MQ for z/OS print log map utility program (`CSQJU004`) runs as a stand-alone utility to list the following information:
- Log data set name and log RBA association for both copies of all active and archive log data sets. If dual logging is not active, there is only one copy of the data sets.
- Active log data sets available for new log data.
- Contents of the queue of checkpoint records in the bootstrap data set (BSDS).
- Contents of the archive log command history record.
- System and utility time stamps.

The log print utility

The log print utility program (`CSQ1LOGP`) is run as a stand-alone utility. You can run the utility specifying:

- A bootstrap data set (BSDS)
- Active logs (with no BSDS)
- Archive logs (with no BSDS)

The queue-sharing group utility

The queue-sharing group utility program (`CSQ5PQSG`) runs as a stand-alone utility to set up Db2 tables and perform other Db2 tasks required for queue-sharing groups.

The active log preformat utility

The active log preformat utility (`CSQJUFMT`) formats active log data sets before they are used by a queue manager. If the active log data sets are preformatted by the utility, log write performance is improved on the queue manager's first pass through the active logs.

The dead-letter queue handler utility

The dead-letter queue handler utility program (`CSQUDLQH`) runs as a stand-alone utility. It checks messages that are on the dead-letter queue and processes them according to a set of rules that you supply to the utility.

The CSQUTIL utility

The CSQUTIL utility program is provided with IBM MQ for z/OS to help you perform backup, restoration, and reorganization tasks, and to issue commands and process object definitions.

For more information about the CSQUTIL utility program, see IBM MQ utility program (CSQUTIL). By using this utility program, you can invoke the following functions:

COMMAND
: To issue MQSC commands, to record object definitions, and to make client-channel definition files.

COPY
: To read the contents of a named IBM MQ for z/OS message queue or the contents of all the queues of a named page set, and put them into a sequential file and retain the original queue.

COPYPAGE
: To copy whole page sets to larger page sets.

EMPTY
: To delete the contents of a named IBM MQ for z/OS message queue or the contents of all the queues of a named page set, retaining the definitions of the queues.

FORMAT
: To format IBM MQ for z/OS page sets.

LOAD
: To restore the contents of a named IBM MQ for z/OS message queue or the contents of all the queues of a named page set from a sequential file created by the COPY function.

PAGEINFO
: To extract page set information from one or more page sets.

RESETPAGE
: To copy whole page sets to other page set data sets and reset the log information in the copy.

SCOPY
To copy the contents of a queue to a data set while the queue manager is offline.

SDEFS
To produce a set of define commands for objects while the queue manager is offline.

SLOAD
To restore messages from the destination data set of an earlier COPY or SCOPY operation. SLOAD processes a single queue.

SWITCH
To switch or query the transmission queue associated with cluster-sender channels.

XPARM
To convert a channel initiator parameter load module into queue manager attributes (for migration purposes).

Operating IBM MQ for z/OS

Use these basic procedures to operate IBM MQ for z/OS.

You can also perform the operations described in this section using the IBM MQ Explorer, which is distributed with IBM MQ for Windows, IBM MQ for Linux (x86 and x86-64 platforms) and SupportPac MS0T. For more information, see "Administration using the MQ Explorer" on page 56 and IBM Support & downloads.

This section contains information about the following topics:

Issuing queue manager commands

You can issue IBM MQ control commands from a z/OS console or with the utility program CSQUTIL. Commands can use command prefix string (CPF) to indicate which IBM MQ subsystem processes the command.

You can control most of the operational environment of IBM MQ using the IBM MQ commands. IBM MQ for z/OS supports both the MQSC and PCF types of these commands. This topic describes how to specify attributes using MQSC commands, and so it refers to those commands and attributes using their MQSC command names, rather than their PCF names. For details of the syntax of the MQSC commands, see The MQSC commands. For details of the syntax of the PCF commands, see "Using Programmable Command Formats" on page 7. If you are a suitably authorized user, you can issue IBM MQ commands from:

- The initialization input data sets (described in "Initialization commands" on page 263).
- A z/OS console, or equivalent, such as SDSF
- The z/OS master get command routine, MGCRE (SVC 34)
- The IBM MQ utility, CSQUTIL (described in IBM MQ utility program.)
- A user application, which can be:
  - A CICS program
  - A TSO program
  - A z/OS batch program
  - An IMS program

  See "Writing programs to administer IBM MQ" on page 287 for information about this.

Much of the functionality of these commands is provided in a convenient way by the operations and control panels, accessible from TSO and ISPF, and described in "Introducing the operations and control panels" on page 272.

For further information, see

- "Issuing commands from a z/OS console or its equivalent"
  - Command prefix strings
  - Using the z/OS console to issue commands
  - Command responses
- Issuing commands from the utility program CSQUTIL

Issuing commands from a z/OS console or its equivalent

You can issue all IBM MQ commands from a z/OS console or its equivalent. You can also issue IBM MQ commands from anywhere where you can issue z/OS commands, such as SDSF or by a program using the MGCRE macro.

The maximum amount of data that can be displayed as a result of a command typed in at the console is 32 KB.

Note:

1. You cannot issue IBM MQ commands using the IMS/SSR command format from an IMS terminal. This function is not supported by the IMS adapter.
2. The input field provided by SDSF might not be long enough for some commands, particularly those commands for channels.

Command prefix strings

Each IBM MQ command must be prefixed with a command prefix string (CPF), as shown in Figure 39.

Because more than one IBM MQ subsystem can run under z/OS, the CPF is used to indicate which IBM MQ subsystem processes the command. For example, to start the queue manager for a subsystem called CSQ1, where CPF is ' `+CSQ1` ', you issue the command `+CSQ1` START QMGR from the operator console. This CPF must be defined in the subsystem name table (for the subsystem CSQ1). This is described in Defining command prefix strings (CPFs). In the examples, the string ' `+CSQ1` ' is used as the command prefix.

Using the z/OS console to issue commands

You can type simple commands from the z/OS console, for example, the DISPLAY command in Figure 39. However, for complex commands or for sets of commands that you issue frequently, the other methods of issuing commands are better.

```
+CSQ1 DISPLAY QUEUE(TRANSMIT.QUEUE.PROD) TYPE(QLOCAL)
```

Figure 39. Issuing a DISPLAY command from the z/OS console

Command responses

Direct responses to commands are sent to the console that issued the command. IBM MQ supports the *Extended Console Support* (EMCS) function available in z/OS, and therefore consoles with 4 byte IDs can be used. Additionally, all commands except START QMGR and STOP QMGR support the use of Command and Response Tokens (CARTs) when the command is issued by a program using the MGCRE macro.

Issuing commands from the utility program CSQUTIL

You can issue commands from a sequential data set using the COMMAND function of the utility program CSQUTIL. This utility transfers the commands, as messages, to the *system-command input queue* and waits for the response, which is printed together with the original commands in SYSPRINT. For details of this, see IBM MQ utility program.

Starting and stopping a queue manager:

Use this topic as an introduction to stopping and starting a queue manager.

This section describes how to start and stop a queue manager. It contains information about the following topics:
- "Before you start IBM MQ"
- "Starting a queue manager"
- "Stopping a queue manager" on page 270

Starting and stopping a queue manager is relatively straightforward. When a queue manager stops under normal conditions, its last action is to take a termination checkpoint. This checkpoint, and the logs, give the queue manager the information it needs to restart.

This section contains information about the START and STOP commands, and contains a brief overview of start-up after an abnormal termination has occurred.

Before you start IBM MQ

After you have installed IBM MQ, it is defined as a formal z/OS subsystem. This message appears during any initial program load (IPL) of z/OS:

```
CSQ3110I +CSQ1 CSQ3UR00 - SUBSYSTEM ssnm INITIALIZATION COMPLETE
```

where *ssnm* is the IBM MQ subsystem name.

From now on, you can start the queue manager for that subsystem *from any z/OS console that has been authorized to issue system control commands* ; that is, a z/OS SYS command group. You must issue the START command from the authorized console, you cannot issue it through JES or TSO.

If you are using queue-sharing groups, you must start RRS first, and then Db2, before you start the queue manager.

Starting a queue manager

You start a queue manager by issuing a START QMGR command. However, you cannot successfully use the START command unless you have appropriate authority. See the Setting up security on z/OS for information about IBM MQ security. Figure 40 shows examples of the START command. (Remember that you must prefix an IBM MQ command with a command prefix string (CPF).)

```
+CSQ1  START QMGR

+CSQ1  START QMGR PARM(NEWLOG)
```

Figure 40. Starting the queue manager from a z/OS console. The second example specifies a system parameter module name.

See START QMGR for information about the syntax of the START QMGR command.

You cannot run the queue manager as a batch job or start it using a z/OS command START. These methods are likely to start an address space for IBM MQ that then ends abnormally. Nor can you start a queue manager from the CSQUTIL utility program or a similar user application.

You can, however, start a queue manager from an APF-authorized program by passing a START QMGR command to the z/OS MGCRE (SVC 34) service.

If you are using queue-sharing groups, the associated Db2 systems and RRS must be active when you start the queue manager.

Start options

When you start a queue manager, a system parameter module is loaded. You can specify the name of the system parameter module in one of two ways:

- With the PARM parameter of the /cpf START QMGR command, for example
  ```
  /cpf START QMGR PARM(CSQ1ZPRM)
  ```
- With a parameter in the startup procedure, for example, code the JCL EXEC statement as
  ```
  //MQM  EXEC PGM=CSQYASCP,PARM='ZPARM(CSQ1ZPRM)'
  ```

A system parameter module provides information specified when the queue manager was customized.

You can also use the ENVPARM option to substitute one or more parameters in the JCL procedure for the queue manager.

For example, you can update your queue manager startup procedure, so that the DDname CSQINP2 is a variable. This means that you can change the CSQINP2 DDname without changing the startup procedure. This is useful for implementing changes, providing backouts for operators, and queue manager operations.

Suppose your start-up procedure for queue manager CSQ1 looked like Figure 41.

```
//CSQ1MSTR PROC INP2=NORM
//MQMESA  EXEC PGM=CSQYASCP
//STEPLIB DD  DISP=SHR,DSN=thlqual.SCSQANLE
//     DD  DISP=SHR,DSN=thlqual.SCSQAUTH
//     DD  DISP=SHR,DSN=db2qual.SDSNLOAD
//BSDS1  DD  DISP=SHR,DSN=myqual.BSDS01
//BSDS2  DD  DISP=SHR,DSN=myqual.BSDS02
//CSQP0000 DD  DISP=SHR,DSN=myqual.PSID00
//CSQP0001 DD  DISP=SHR,DSN=myqual.PSID01
//CSQP0002 DD  DISP=SHR,DSN=myqual.PSID02
//CSQP0003 DD  DISP=SHR,DSN=myqual.PSID03
//CSQINP1 DD  DISP=SHR,DSN=myqual.CSQINP(CSQ1INP1)
//CSQINP2 DD  DISP=SHR,DSN=myqual.CSQINP(CSQ1&INP2.)
//CSQOUT1 DD  SYSOUT=*
//CSQOUT2 DD  SYSOUT=*
```

Figure 41. Sample start-up procedure

If you then start your queue manager with the command:

```
+CSQ1  START QMGR
```

the CSQINP2 used is a member called CSQ1NORM.

However, suppose you are putting a new suite of programs into production so that the next time you start queue manager CSQ1, the CSQINP2 definitions are to be taken from member CSQ1NEW. To do this, you would start the queue manager with this command:

```
+CSQ1  START QMGR ENVPARM('INP2=NEW')
```

and CSQ1NEW would be used instead of CSQ1NORM. Note: z/OS limits the `KEYWORD=value` specifications for symbolic parameters (as in `INP2=NEW`) to 255 characters.

Starting after an abnormal termination

IBM MQ automatically detects whether restart follows a normal shutdown or an abnormal termination.

Starting a queue manager after it ends abnormally is different from starting it after the STOP QMGR command has been issued. After STOP QMGR, the system finishes its work in an orderly way and takes a termination checkpoint before stopping. When you restart the queue manager, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

However, if the queue manager ends abnormally, it terminates without being able to finish its work or take a termination checkpoint. When you restart a queue manager after an abend, it refreshes its knowledge of its status at termination using information in the log, and notifies you of the status of various tasks. Normally, the restart process resolves all inconsistent states. But, in some cases, you must take specific steps to resolve inconsistencies.

User messages on start-up

When you start a queue manager successfully, the queue manager produces a set of startup messages.

Stopping a queue manager

Before stopping a queue manager, all IBM MQ-related write-to-operator-with-reply (WTOR) messages must receive replies, for example, getting log requests. Each command in Figure 42 terminates a running queue manager.

```
+CSQ1  STOP QMGR

+CSQ1  STOP QMGR MODE(QUIESCE)

+CSQ1  STOP QMGR MODE(FORCE)

+CSQ1  STOP QMGR MODE(RESTART)
```

Figure 42. Stopping a queue manager

The command STOP QMGR defaults to STOP QMGR MODE(QUIESCE).

In QUIESCE mode, IBM MQ does not allow any new connection threads to be created, but allows existing threads to continue; it terminates only when all threads have ended. Applications can request to be notified in the event of the queue manager quiescing. Therefore, use the QUIESCE mode where possible so that applications that have requested notification have the opportunity to disconnect. See What happens during termination for details.

If the queue manager does not terminate in a reasonable time in response to a STOP QMGR MODE(QUIESCE) command, use the DISPLAY CONN command to determine whether any connection threads exist, and take the necessary steps to terminate the associated applications. If there are no threads, issue a STOP QMGR MODE(FORCE) command.

The STOP QMGR MODE(QUIESCE) and STOP QMGR MODE(FORCE) commands deregister IBM MQ from the MVS Automatic Restart Manager (ARM), preventing ARM from restarting the queue manager automatically. The STOP QMGR MODE(RESTART) command works in the same way as the STOP QMGR MODE(FORCE) command, except that it does not deregister IBM MQ from ARM. This means that the queue manager is eligible for immediate automatic restart.

If the IBM MQ subsystem is not registered with ARM, the STOP QMGR MODE(RESTART) command is rejected and the following message is sent to the z/OS console:

```
CSQY205I ARM element arm-element is not registered
```

If this message is not issued, the queue manager is restarted automatically. For more information about ARM, see "Using the z/OS Automatic Restart Manager (ARM)" on page 349.

Only cancel the queue manager address space if STOP QMGR MODE(FORCE) does not terminate the queue manager.

If a queue manager is stopped by either canceling the address space or by using the command STOP QMGR MODE(FORCE), consistency is maintained with connected CICS or IMS systems. Resynchronization of resources is started when a queue manager restarts and is completed when the connection to the CICS or IMS system is established.

Note: When you stop your queue manager, you might find message IEF352I is issued. z/OS issues this message if it detects that failing to mark the address space as unusable would lead to an integrity exposure. You can ignore this message.

Stop messages

After issuing a STOP QMGR command, you get the messages CSQY009I and CSQY002I, for example:

```
CSQY009I +CSQ1 ' STOP QMGR' COMMAND ACCEPTED FROM
USER(userid), STOP MODE(FORCE)
CSQY002I +CSQ1 QUEUE MANAGER STOPPING
```

Where `userid` is the user ID that issued the STOP QMGR command, and the MODE parameter depends on that specified in the command.

When the STOP command has completed successfully, the following messages are displayed on the z/OS console:

```
CSQ9022I +CSQ1 CSQYASCP ' STOP QMGR' NORMAL COMPLETION
CSQ3104I +CSQ1 CSQ3EC0X - TERMINATION COMPLETE
```

If you are using ARM, and you did not specify MODE(RESTART), the following message is also displayed:

```
CSQY204I +CSQ1 ARM DEREGISTER for element arm-element type
arm-element-type successful
```

You cannot restart the queue manager until the following message has been displayed:

```
CSQ3100I +CSQ1 CSQ3EC0X - SUBSYSTEM ssnm READY FOR START COMMAND
```

Introducing the operations and control panels

You can use the IBM MQ operations and control panels to perform administration tasks on IBM MQ objects. Use this topic as an introduction to the commands, and control panels.

You use these panels for defining, displaying, altering, or deleting IBM MQ objects. Use the panels for day-to-day administration and for making small changes to objects. If you are setting up or changing many objects, use the COMMAND function of the CSQUTIL utility program.

The operations and control panels support the controls for the channel initiator (for example, to start a channel or a TCP/IP listener), for clustering, and for security. They also enable you to display information about threads and page set usage.

The panels work by sending MQSC type IBM MQ commands to a queue manager, through the system command input queue.

Note:

1. The z/OS IBM MQ operations and controls panels (CSQOREXX) might not support all new function and parameters added from version 7 onwards. For example, there are no panels for the direct manipulation of topic objects or subscriptions.

   Using one of the following supported mechanisms allows you to administer publish/subscribe definitions and other system controls that are not directly available from other panels:

   a. IBM MQ Explorer
   b. z/OS console
   c. Programmable Command Format (PCF) messages
   d. COMMAND function of CSQUTIL

   Note that the generic **Command** action in the CSQOREXX panels allows you to issue any valid MQSC command, including SMDS related commands. You can use all the commands that the COMMAND function of CSQUTIL issues.
2. You cannot issue the IBM MQ commands directly from the command line in the panels.
3. To use the operations and control panels, you must have the correct security authorization; this is described in the User IDs for command security and command resource security.
4. You cannot provide a user ID and password using CSQUTIL, or the CSQOREXX panels. Instead, if you user ID has UPDATE authority to the BATCH profile in MQCONN, you can bypass the **CHCKLOCL**(*REQUIRED* setting. See Using **CHCKLOCL** on locally bound applications for more information.

Invocation and rules for the operations and control panels:

You can control IBM MQ and issue control commands through the ISPF panels.

If the ISPF/PDF primary options menu has been updated for IBM MQ, you can access the IBM MQ operations and control panels from that menu. For details about updating the menu, see the Task 20: Set up the operations and control panels.

You can access the IBM MQ operations and control panels from the TSO command processor panel (typically option 6 on the ISPF/PDF primary options menu). The name of the exec that you run to do this is CSQOREXX. It has two parameters; `thlqual` is the high-level qualifier for the IBM MQ libraries to be used, and `langletter` is the letter identifying the national language libraries to be used (for example, E for U.S. English). The parameters can be omitted if the IBM MQ libraries are permanently installed in your ISPF setup. Alternatively, you can issue CSQOREXX from the TSO command line.

These panels are designed to be used by operators and administrators with a minimum of formal training. Read these instructions with the panels running and try out the different tasks suggested.

Note: While using the panels, temporary dynamic queues with names of the form SYSTEM.CSQOREXX.* are created.

Rules for the operations and control panels

See Rules for naming IBM MQ objects about the general rules for IBM MQ character strings and names. However, there are some rules that apply only to the operations and control panels:

- Do not enclose strings, for example descriptions, in single or double quotation marks.
- If you include an apostrophe or quotation mark in a text field, you do not have to repeat it or add an escape character. The characters are saved exactly as you type them; for example:

  ```
  This is Maria's queue
  ```

  The panel processor doubles them for you to pass them to IBM MQ. However, if it has to truncate your data to do this, it does so.
- You can use uppercase or lowercase characters in most fields, and they are folded to uppercase characters when you press Enter. The exceptions are:
  - Storage class names and coupling facility structure names, which must start with uppercase A through Z and be followed by uppercase A through Z or numeric characters.
  - Certain fields that are not translated. These include:
    - Application ID
    - Description
    - Environment data
    - Object names (but if you use a lowercase object name, you might not be able to enter it at a z/OS console)
    - Remote system name
    - Trigger data
    - User data
- In names, leading blanks and leading underscores are ignored. Therefore, you cannot have object names beginning with blanks or underscores.
- Underscores are used to show the extent of blank fields. When you press Enter, trailing underscores are replaced by blanks.

- Many description and text fields are presented in multiple parts, each part being handled by IBM MQ independently. This means that trailing blanks *are retained* and the text is not contiguous.

Blank fields

When you specify the **Define** action for an IBM MQ object, each field on the define panel contains a value. See the general help (extended help) for the display panels for information about where IBM MQ gets the values. If you type over a field with blanks, and blanks are not allowed, IBM MQ puts the installation default value in the field or prompts you to enter the required value.

When you specify the **Alter** action for an IBM MQ object, each field on the alter panel contains the current value for that field. If you type over a field with blanks, and blanks are not allowed, the value of that field is left unchanged.

Objects and actions:

The operations and control panels offer you many different types of object and a number of actions that you can perform on them.

The actions are listed on the initial panel and enable you to manipulate the objects and display information about them. These objects include all the IBM MQ objects, together with some extra ones. The objects fall into the following categories.
- Queues, processes, authentication information objects, namelists, storage classes and CF structures
- Channels
- Cluster objects
- Queue manager and security
- Connections
- System

Refer to Actions for a cross reference table of the actions which can be taken with the IBM MQ objects.

Queues, processes, authentication information objects, namelists, storage classes and CF structures

These are the basic IBM MQ objects. There can be many of each type. They can be listed, listed with filter, defined, and deleted, and have attributes that can be displayed and altered, using the LIST or DISPLAY, LIST with FILTER, DEFINE LIKE, MANAGE, and ALTER actions. (Objects are deleted using the MANAGE action.)

This category consists of the following objects:

QLOCAL	Local queue
QREMOTE	Remote queue
QALIAS	Alias queue for indirect reference to a queue
QMODEL	Model queue for defining queues dynamically
QUEUE	Any type of queue
QSTATUS	Status of a local queue
PROCESS	Information about an application to be started when a trigger event occurs
AUTHINFO	Authentication information: definitions required to perform Certificate Revocation List (CRL) checking using LDAP servers
NAMELIST	List of names, such as queues or clusters
STGCLASS	Storage class
CFSTRUCT	coupling facility (CF) structure
CFSTATUS	Status of a CF structure

Channels

Channels are used for distributed queuing. There can be many of each type, and they can be listed, listed with filter, defined, deleted, displayed, and altered. They also have other functions available using the START, STOP and PERFORM actions. PERFORM provides reset, ping, and resolve channel functions.

This category consists of the following objects:

CHANNEL	Any type of channel
SENDER	Sender channel
SERVER	Server channel
RECEIVER	Receiver channel
REQUESTER	Requester channel
CLUSRCVR	Cluster-receiver channel
CLUSSDR	Cluster-sender channel
SVRCONN	Server-connection channel
CLNTCONN	Client-connection channel
CHSTATUS	Status of a channel connection

Cluster objects

Cluster objects are created automatically for queues and channels that belong to a cluster. The base queue and channel definitions can be on another queue manager. There can be many of each type, and names can be duplicated. They can be listed, listed with filter, and displayed. PERFORM, START, and STOP are also available through the LIST actions.

This category consists of the following objects:

CLUSQ	Cluster queue, created for a queue that belongs to a cluster
CLUSCHL	Cluster channel, created for a channel that belongs to a cluster
CLUSQMGR	Cluster queue manager, the same as a cluster channel but identified by its queue manager name

Cluster channels and cluster queue managers do have the PERFORM, START and STOP actions, but only indirectly through the DISPLAY action.

Queue manager and security

Queue manager and security objects have a single instance. They can be listed, and have attributes that can be displayed and altered (using the LIST or DISPLAY, and ALTER actions), and have other functions available using the PERFORM action.

This category consists of the following objects:

MANAGER	Queue manager: the PERFORM action provides suspend and resume cluster functions
SECURITY	Security functions: the PERFORM action provides refresh and reverify functions

Connection

Connections can be listed, listed with filter and displayed.

This category consists only of the connection object, CONNECT.

System

A collection of other functions. This category consists of the following objects:

SYSTEM	System functions
CONTROL	Synonym for SYSTEM

The functions available are:

LIST or DISPLAY	Display queue-sharing group, distributed queuing, page set, or data set usage information.
PERFORM	Refresh or reset clustering
START	Start the channel initiator or listeners
STOP	Stop the channel initiator or listeners

Actions

The actions that you can perform for each type of object are shown in the following table:

Table 13. Valid operations and control panel actions for IBM MQ objects

Object	Alter	Define like	Manage (1)	List or Display	List with Filter	Perform	Start	Stop
AUTHINFO	X	X	X	X	X			
CFSTATUS				X				
CFSTRUCT	X	X	X	X	X			
CHANNEL	X	X	X	X	X	X	X	X
CHSTATUS				X	X			
CLNTCONN	X	X	X	X	X			
CLUSCHL				X	X	X(2)	X(2)	X(2)
CLUSQ				X	X			
CLUSQMGR				X	X	X(2)	X(2)	X(2)
CLUSRCVR	X	X	X	X	X	X	X	X
CLUSSDR	X	X	X	X	X	X	X	X
CONNECT				X	X			
CONTROL				X		X	X	X
MANAGER	X			X		X		
NAMELIST	X	X	X	X	X			
PROCESS	X	X	X	X	X			
QALIAS	X	X	X	X	X			
QLOCAL	X	X	X	X	X			

Table 13. Valid operations and control panel actions for IBM MQ objects (continued)

Object	Alter	Define like	Manage (1)	List or Display	List with Filter	Perform	Start	Stop
QMODEL	X	X	X	X	X			
QREMOTE	X	X	X	X	X			
QSTATUS				X	X			
QUEUE	X	X	X	X	X			
RECEIVER	X	X	X	X	X	X	X	X
REQUESTER	X	X	X	X	X	X	X	X
SECURITY	X			X		X		
SENDER	X	X	X	X	X	X	X	X
SERVER	X	X	X	X	X	X	X	X
SVRCONN	X	X	X	X	X		X	X
STGCLASS	X	X	X	X	X			
SYSTEM				X		X	X	X

Note:
1. Provides **Delete** and other functions
2. Using the **List or Display** action

Object dispositions:

You can specify the *disposition* of the object with which you need to work. The disposition signifies where the object **definition** is kept, and how the object behaves.

The disposition is significant only if you are working with any of the following object types:
- queues
- channels
- processes
- namelists
- storage classes
- authentication information objects

If you are working with other object types, the disposition is disregarded.

Permitted values are:

Q	QMGR. The object definitions are on the page set of the queue manager and are accessible only by the queue manager.
C	COPY. The object definitions are on the page set of the queue manager and are accessible only by the queue manager. They are local copies of objects defined as having a disposition of GROUP.
P	PRIVATE. The object definitions are on the page set of the queue manager and are accessible only by the queue manager. The objects have been defined as having a disposition of QMGR or COPY.
G	GROUP. The object definitions are in the shared repository, and are accessible by all queue managers in the queue-sharing group.
S	SHARED. This disposition applies only to local queues. The queue definitions are in the shared repository, and are accessible by all queue managers in the queue-sharing group.

A ALL. If the action queue manager is either the target queue manager, or *, objects of **all** dispositions are included; otherwise, objects of QMGR and COPY dispositions only are included. This is the default.

Selecting a queue manager, defaults, and levels using the ISPF control panel:

You can use the CSQOREXX exec in ISPF to control your queue managers.

While you are viewing the initial panel, you are not connected to any queue manager. However, as soon as you press Enter, you are connected to the queue manager, or a queue manager in the queue-sharing group named in the **Connect name** field. You can leave this field blank; this means that you are using the default queue manager for batch applications. This is defined in CSQBDEFV (see Task 19: Set up Batch, TSO, and RRS adapters for information about this).

Use the **Target queue manager** field to specify the queue manager where the actions you request are to be performed. If you leave this field blank, it defaults to the queue manager specified in the **Connect name** field. You can specify a target queue manager that is not the one you connect to. In this case, you would normally specify the name of a remote queue manager object that provides a queue manager alias definition (the name is used as the *ObjectQMgrName* when opening the command input queue). To do this, you must have suitable queues and channels set up to access the remote queue manager.

The **Action queue manager** allows you to specify a queue manager that is in the same queue-sharing group as the queue manager specified in the **Target queue manager** field to be the queue manager where the actions you request are to be performed. If you specify * in this field, the actions you request are performed on all queue managers in the queue-sharing group. If you leave this field blank, it defaults to the value specified in the **Target queue manager** field. The **Action queue manager** field corresponds to using the CMDSCOPE command modifier described in The MQSC commands.

Queue manager defaults

If you leave any queue manager fields blank, or choose to connect to a queue-sharing group, a secondary window opens when you press Enter. This window confirms the names of the queue managers you will be using. Press Enter to continue. When you return to the initial panel after having made some requests, you find fields completed with the actual names.

Queue manager levels

The Operations and Control panels work satisfactorily only with queue managers that are running on z/OS, and with command levels that match that of the panels, currently 710 or 800.

If these conditions are not met, it is likely that actions work only partially, incorrectly, or not at all, and that the replies from the queue manager are not recognized.

If the action queue manager is not at command level 800, some fields are not displayed, and some values cannot be entered. A few objects and actions are disallowed. In such cases, a secondary window opens asking for you to confirm that you want to proceed.

Using the function keys and command line with the ISPF control panels:

To use the panels, you must use the function keys or enter the equivalent commands in the ISPF control panel command area.

- Function keys
  - Processing your actions
  - "Displaying IBM MQ user messages"
  - Canceling your actions
  - Getting help
- Using the command line

Function keys

The function keys have special settings for IBM MQ. (This means that you cannot use the ISPF default values for the function keys; if you have previously used the KEYLIST OFF ISPF command anywhere, you must type KEYLIST ON in the command area of any operations and control panel and then press Enter to enable the IBM MQ settings.)

These function key settings can be displayed on the panels, as shown in Figure 43 on page 280. If the settings are not shown, type PFSHOW in the command area of any operations and control panel and then press Enter. To remove the display of the settings, use the command PFSHOW OFF.

The function key settings in the operations and control panels conform to CUA standards. Although you can change the key setting through normal ISPF procedures (such as the KEYLIST utility), you are not recommended to do so.

Note: Using the PFSHOW and KEYLIST commands affects any other logical ISPF screens that you have, and their settings remain when you leave the operations and control panels.

Processing your actions

Press Enter to carry out the action requested on a panel. The information from the panel is sent to the queue manager for processing.

Each time you press Enter in the panels, IBM MQ generates one or more operator messages. If the operation was successful, you get confirmation message CSQ9022I, otherwise you get some error messages.

Displaying IBM MQ user messages

Press function key F10 in any panel to see the IBM MQ user messages.

Canceling your actions

On the initial panel, both F3 and F12 exit the operations and control panels and return you to ISPF. No information is sent to the queue manager.

On any other panel, press function keys F3 or F12 to leave the current panel **ignoring any data you have typed since last pressing Enter**. Again, no information is sent to the queue manager.

- F3 takes you straight back to the initial panel.
- F12 takes you back to the previous panel.

Getting help

Each panel has help panels associated with it. The help panels use the ISPF protocols:

- Press function key F1 on any panel to see general help (extended help) about the task.
- Press function key F1 with the cursor on any field to see specific help about that field.
- Press function key F5 from any field help panel to get the general help.

- Press function key F3 to return to the base panel, that is, the panel from which you pressed function key F1.
- Press function key F6 from any help panel to get help about the function keys.

If the help information carries on into a second or subsequent pages, a **More** indicator is displayed in the upper right of the panel. Use these function keys to navigate through the help pages:

- F11 to get to the next help page (if there is one).
- F10 to get back to the previous help page (if there is one).

Using the command line

You never need to use the command line to issue the commands used by the operations and control panels because they are available from function keys. The command line is provided to allow you to enter normal ISPF commands (like PFSHOW).

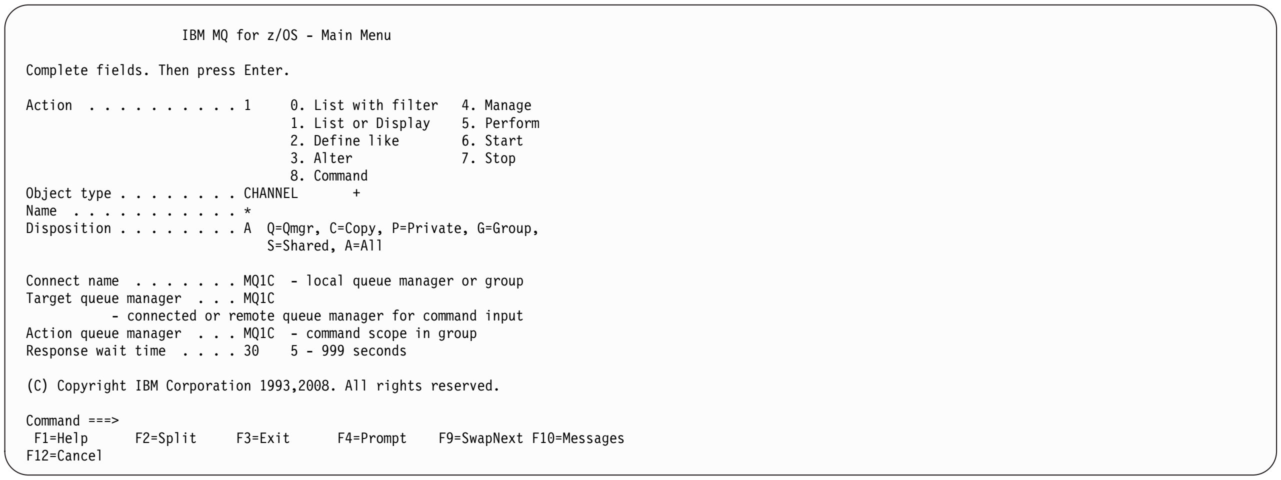
The ISPF command PANELID ON displays the name of the current CSQOREXX panel.

The command line is initially displayed in the default position at the bottom of the panels, regardless of what ISPF settings you have. You can use the SETTINGS ISPF command from any of the operations and control panels to change the position of the command line. The settings are remembered for subsequent sessions with the operations and control panels.

Using the operations and control panels

Use this topic to investigate the initial control panel displayed from CSQOREXX

Figure 43 shows the panel that is displayed when you start a panel session.



```
                    IBM MQ for z/OS - Main Menu

Complete fields. Then press Enter.

Action  . . . . . . . . . . 1     0. List with filter   4. Manage
                                  1. List or Display    5. Perform
                                  2. Define like        6. Start
                                  3. Alter              7. Stop
                                  8. Command
Object type . . . . . . . . CHANNEL       +
Name  . . . . . . . . . . . *
Disposition . . . . . . . . A  Q=Qmgr, C=Copy, P=Private, G=Group,
                               S=Shared, A=All

Connect name  . . . . . . . MQ1C  - local queue manager or group
Target queue manager  . . . MQ1C
         - connected or remote queue manager for command input
Action queue manager  . . . MQ1C  - command scope in group
Response wait time  . . . . 30    5 - 999 seconds

(C) Copyright IBM Corporation 1993,2008. All rights reserved.

Command ===>
 F1=Help      F2=Split     F3=Exit      F4=Prompt    F9=SwapNext F10=Messages
F12=Cancel
```

Figure 43. The IBM MQ operations and control initial panel

From this panel you can perform actions such as:

- Choose the local queue manager you want and whether you want the commands issued on that queue manager, on a remote queue manager, or on another queue manager in the same queue-sharing group as the local queue manager. Over type the queue manager name if you need to change it.
- Select the action you want to perform by typing in the appropriate number in the **Action** field.
- Specify the object type that you want to work with. Press function key F1 for help about the object types if you are not sure what they are.
- Specify the disposition of the object type that you want to work with.

- Display a list of objects of the type specified. Type in an asterisk (*) in the **Name** field and press Enter to display a list of objects (of the type specified) that have already been defined on the action queue manager. You can then select one or more objects to work with in sequence. All the actions are available from the list.

Note: You are recommended to make choices that result in a list of objects being displayed, and then work from that list. Use the **Display** action, because that is allowed for all object types.

Using the Command Facility

Use the editor to enter or amend MQSC commands to be passed to the queue manager.

From the primary panel, CSQOPRIA, select option **8 Command**, to start the Command Facility.

You are presented with an edit session of a sequential file, *prefix*.CSQUTIL.COMMANDS, used as input to the CSQUTIL COMMAND function; see Issuing commands to IBM MQ.

You do not need to prefix commands with the command prefix string (CPF).

You can continue MQSC commands on subsequent lines by terminating the current line with the continuation characters **+** or **-**. Alternatively, use line edit mode to provide long MQSC commands or the values of long attribute values within the command.

line edit

To use line edit, move the cursor to the appropriate line in the edit panel and use **F4** to display a single line in a scrollable panel. A single line can be up to 32 760 bytes of data.

To leave line edit:

- **F3 exit** saves changes made to the line and exits
- **F12 cancel** returns to the edit panel discarding changes made to the line.

To discard changes made in the edit session, use **F12 cancel** to terminate the edit session leaving the contents of the file unchanged. Commands are not executed.

Executing commands

When you have finished entering MQSC commands, terminate the edit session with **F3 exit** to save the contents of the file and invoke CSQUTIL to pass the commands to the queue manager. The output from command processing is held in file *prefix*.CSQUTIL.OUTPUT. An edit session opens automatically on this file so that you can view the responses. Press **F3 exit** to exit this session and return to the main menu.

Working with IBM MQ objects

Many of the tasks described in this documentation involve manipulating IBM MQ objects. The object types are queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects.

- Defining simple queue objects
- Defining other types of objects
- Working with object definitions
- Working with namelists

Defining simple queue objects

To define a new object, use an existing definition as the basis for it. You can do this in one of three ways:

- By selecting an object that is a member of a list displayed as a result of options selected on the initial panel. You then enter action type 2 (**Define like**) in the action field next to the selected object. Your new object has the attributes of the selected object, except the disposition. You can then change any attributes in your new object as you require.
- On the initial panel, select the **Define like** action type, enter the type of object that you are defining in the **Object type** field, and enter the name of a specific existing object in the **Name** field. Your new object has the same attributes as the object you named in the **Name** field, except the disposition. You can then change any attributes in your new object definition as you require.
- By selecting the **Define like** action type, specifying an object type and then leaving the **Name** field blank. You can then define your new object and it has the default attributes defined for your installation. You can then change any attributes in your new object definition as you require.

Note: You do not enter the name of the object you are defining on the initial panel, but on the **Define** panel you are presented with.

The following example demonstrates how to define a local queue using an existing queue as a template.

Defining a local queue

To define a local queue object from the operations and control panels, use an existing queue definition as the basis for your new definition. There are several panels to complete. When you have completed all the panels and you are satisfied that the attributes are correct, press Enter to send your definition to the queue manager, which then creates the actual queue.

Use the **Define like** action either on the initial panel or against an object entry in a list displayed as a result of options selected on the initial panel.

For example, starting from the initial panel, complete these fields:

Action	2 (Define like)
Object type	QLOCAL
Name	QUEUE.YOU.LIKE. This is the name of the queue that provides the attributes for your new queue.

Press Enter to display the **Define a Local Queue** panel. The queue name field is blank so that you can supply the name for the new queue. The description is that of the queue upon which you are basing this new definition. Over type this field with your own description for the new queue.

The values in the other fields are those of the queue upon which you are basing this new queue, except the disposition. You can over type these fields as you require. For example, type `Y` in the **Put enabled** field (if it is not already Y) if suitably authorized applications can put messages on this queue.

You get field help by moving the cursor into a field and pressing function key F1. Field help provides information about the values that can be used for each attribute.

When you have completed the first panel, press function key F8 to display the second panel.

Hints:

1. Do *not* press Enter at this stage, otherwise the queue will be created before you have a chance to complete the remaining fields. (If you do press Enter prematurely, do not worry; you can always alter your definition later on.)
2. Do not press function keys F3 or F12, or the data you typed will be lost.

Press function key F8 repeatedly to see and complete the remaining panels, including the trigger definition, event control, and backout reporting panels.

When your local queue definition is complete

When your definition is complete, press Enter to send the information to the queue manager for processing. The queue manager creates the queue according to the definition you have supplied. If you do not want the queue to be created, press function key F3 to exit and cancel the definition.

Defining other types of objects

To define other types of object, use an existing definition as the base for your new definition as explained in Defining a local queue.

Use the **Define like** action either on the initial panel or against an object entry in a list displayed as a result of options selected on the initial panel.

For example, starting from the initial panel, complete these fields:

Action	2 (Define like)
Object type	QALIAS, NAMELIST, PROCESS, CHANNEL, and other resource objects.
Name	Leave blank or enter the name of an existing object of the same type.

Press Enter to display the corresponding DEFINE panels. Complete the fields as required and then press Enter again to send the information to the queue manager.

Like defining a local queue, defining another type of object generally requires several panels to be completed. Defining a namelist requires some additional work, as described in "Working with namelists" on page 284.

Working with object definitions

When an object has been defined, you can specify an action in the **Action** field, to alter, display, or manage it.

In each case, you can either:

- Select the object you want to work with from a list displayed as a result of options selected on the initial panel. For example, having entered 1 in the **Action** field to display objects, `Queue` in the **Object type** field, and * in the **Name** field, you are presented with a list of all queues defined in the system. You can then select from this list the queue with which you need to work.
- Start from the initial panel, where you specify the object you are working with by completing the **Object type** and **Name** fields.

Altering an object definition

To alter an object definition, specify action 3 and press Enter to see the ALTER panels. These panels are very similar to the DEFINE panels. You can alter the values you want. When your changes are complete, press Enter to send the information to the queue manager.

Displaying an object definition

If you want to see the details of an object without being able to change them, specify action 1 and press Enter to see the DISPLAY panels. Again, these panels are similar to the DEFINE panels except that you cannot change any of the fields. Change the object name to display details of another object.

Deleting an object

To delete an object, specify action 4 (Manage) and the **Delete** action is one of the actions presented on the resulting menu. Select the **Delete** action.

You are asked to confirm your request. If you press function key F3 or F12, the request is canceled. If you press Enter, the request is confirmed and passed to the queue manager. The object you specified is then deleted.

Note: You cannot delete most types of channel object unless the channel initiator is started.

Working with namelists

When working with namelists, proceed as you would for other objects.

For the actions DEFINE LIKE or ALTER, press function key F11 to add names to the list or to change the names in the list. This involves working with the ISPF editor and all the normal ISPF edit commands are available. Enter each name in the namelist on a separate line.

When you use the ISPF editor in this way, the function key settings are the normal ISPF settings, and **not** those used by the other operations and control panels.

If you need to specify lowercase names in the list, specify CAPS(OFF) on the editor panel command line. When you do this, all the namelists that you edit in the future are in lowercase until you specify CAPS(ON).

When you have finished editing the namelist, press function key F3 to end the ISPF edit session. Then press Enter to send the changes to the queue manager.

Attention: If you do not press Enter at this stage but press function key F3 instead, you lose any updates that you have typed in.

Implementing the system using multiple cluster transmission queues

It makes no difference if the channel is used in a single cluster, or an overlapping cluster. When the channel is selected and started, the channel selects the transmission queue depending on the definitions.

About this task

See "Using the automatic definition of queues and switching" if you are using the DEFCLXQ option.

See "Changing your cluster-sender channels using a phased approach" if you are using a staged approach.

Using the automatic definition of queues and switching:

Use this option if you are planning on using the DEFCLXQ option. There will be a queue created for every channel, and every new channel.

Procedure

1. Review the definition of the SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE and change the attributes if required. This queue is defined in member `SCSQPROC(csq4insx)`.
2. Create the SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE model queue.
3. Apply security policies for this model queue, and the SYSTEM.CLUSTER.TRANSMIT.** queues. For z/OS the channel initiator started task user ID needs:
   - Control access to CLASS(MQADMIN) for `ssid.CONTEXT.SYSTEM.CLUSTER.TRANSMIT.channelname`
   - Update access to CLASS(MQQUEUE) for `ssid.SYSTEM.CLUSTER.TRANSMIT.channelname`

Changing your cluster-sender channels using a phased approach:

This process allows you to move to the new cluster-sender channels at various times to suit the needs of your enterprise.

Before you begin

- Identify your business applications, and which channels are used.
- For the queues you use, display the clusters they are in.
- Display the channels to show the connection names, the names of the remote queue managers, and which clusters the channel supports.

About this task

- Create a transmission queue. On z/OS you might want to consider which page set you use for the queue.
- Set up security policy for the queue.
- Change any queue monitoring to include this queue name.
- Decide which channels are to use this transmission queue. The channels should have a similar name, so generic characters ' * ' in the CLCHNAME identify the channel.
- When you are ready to use the new function, alter the transmission queue to specify the name of the channels to use this transmission queue. For example CLUSTER1.TOPARIS, or CLUSTER1.* or *.TOPARIS
- Start the channels

Procedure

1. Use the `DIS CLUSQMGR(xxxx) XMITQ` command to display the cluster sender channels defined in the cluster, where *xxxx* is the name of the remote queue manager.
2. Set up the security profile for the transmission queue and give the queue access to the channel initiator.
3. Define the transmission queue to be used, and specify USAGE(XMITQ) INDXTYPE(CORRELID) SHARE and CLCHNAME(*value*) The channel initiator started task user ID needs the following access:

   ```
   alter class(MQADMIN) ssid.CONTEXT.SYSTEM.CLUSTER.TRANSMIT.channel
   update class(MQQUEUE ssid.SYSTEM.CLUSTER.TRANSMIT.channel
   ```

   and the user ID using the SWITCH command needs the following access:

   ```
   alter cl(MQADMIN) ssid.QUEUE.queuename
   ```
4. Stop and restart the channels.

   The channel change occurs when the channel starts using an MQSC command, or you use CSQUTIL. You can identify which channels need to be restarted using the `SWITCH CHANNEL(*)STATUS` of CSQUTIL

   If you have problems when the channel is started, stop the channel, resolve the problems, and restart the channel.

   Note that you can change the CLCHNAME attribute as often as you need to.

   The value of CLCHNAME used is the one when the channel is started, so you can change the CLCHNAME definition while the channel continues to use the definitions from the time that it started. The channel uses the new definition when it is restarted.

Undoing a change

You need to have a process to backout a change if it the results are not as you expect.

What can go wrong?

If the new transmission queue is not what you expect:

1. Check the CLCHNAME is as you expect
2. Review the job log to check if the switch process has finished. If not, wait and check the new transmission queue of the channel later.

If you are using multiple cluster transmission queues, it is important that you design the transmission queues definitions explicitly and avoid complicated overlapping configuration. In this way, you can make sure that if there are problems, you can go back to the original queues and configuration.

If you encounter problems during the move to using a different transmission queue, you must resolve any problems before you can proceed with the change.

An existing change request must complete before a new change request can be made. For example, you:

1. Define a new transmission queue with a maximum depth of one and there are 10 messages waiting to be sent.
2. Change the transmission queue to specify the channel name in the CLCHNAME parameter.
3. Stop and restart the channel. The attempt to move the messages fails and reports the problems.
4. Change the CLCHNAME parameter on the transmission queue to be blank.
5. Stop and restart the channel. The channel continues to try and complete the original request, so the channel continues to use the new transmission queue.
6. Need to resolve the problems and restart the channel so the moving of messages completes successfully.

Next time the channel is restarted it picks up any changes, so if you had set CLCHNAME to blanks, the channel will not use the specified transmission queue.

In this example, changing the CLCHNAME on the transmission queue to blanks does not necessarily mean that the channel uses the SYSTEM.CLUSTER.TRANSMIT queue, as there might be other transmission queues whose CLCHNAME parameter match the channel name. For example, a generic name, or the queue manager attribute DEFCLXQ might be set to channel, so the channel uses a dynamic queue instead of the SYSTEM.CLUSTER.TRANSMIT queue.

Writing programs to administer IBM MQ

You can write your own application programs to administer a queue manager. Use this topic to understand the requirements for writing your own administration programs.

Start of General-use programming interface information

This set of topics contains hints and guidance to enable you to issue IBM MQ commands from an IBM MQ application program.

Note: In this topic, the MQI calls are described using C-language notation. For typical invocations of the calls in the COBOL, PL/I, and assembler languages, see Function calls manual.

Understanding how it all works

In outline, the procedure for issuing commands from an application program is as follows:

1. Build an IBM MQ command into a type of IBM MQ message called a *request message*. The command can be in MQSC or PCF format.
2. Send (use **MQPUT**) this message to a special queue called the system-command input queue. The IBM MQ command processor runs the command.
3. Retrieve (use **MQGET**) the results of the command as *reply messages* on the reply-to queue. These messages contain the user messages that you need to determine whether your command was successful and, if it was, what the results were.

Then it is up to your application program to process the results.

This set of topics contains:

Preparing queues for administration programs

Administration programs require a number of predefined queues for system command input and receiving responses.

This section applies to commands in the MQSC format. For the equivalent in PCF, see “Using Programmable Command Formats” on page 7.

Before you can issue any **MQPUT** or **MQGET** calls, you must first define, and then open, the queues you are going to use.

Defining the system-command input queue

The system-command input queue is a local queue called SYSTEM.COMMAND.INPUT. The supplied CSQINP2 initialization data set, thlqual.SCSQPROC(CSQ4INSG), contains a default definition for the system-command input queue. For compatibility with IBM MQ on other platforms, an alias of this queue, called SYSTEM.ADMIN.COMMAND.QUEUE is also supplied. See Sample definitions supplied with IBM MQ for more information.

Defining a reply-to queue

You must define a reply-to queue to receive reply messages from the IBM MQ command processor. It can be any queue with attributes that allow reply messages to be put on it. However, for normal operation, specify these attributes:

- USAGE(NORMAL)
- NOTRIGGER (unless your application uses triggering)

Avoid using persistent messages for commands, but if you choose to do so, the reply-to queue must not be a temporary dynamic queue.

The supplied CSQINP2 initialization data set, thlqual.SCSQPROC(CSQ4INSG), contains a definition for a model queue called SYSTEM.COMMAND.REPLY.MODEL. You can use this model to create a dynamic reply-to queue.

Note: Replies generated by the command processor can be up to 15 000 bytes in length.

If you use a permanent dynamic queue as a reply-to queue, your application should allow time for all PUT and GET operations to complete before attempting to delete the queue, otherwise MQRC2055 (MQRC_Q_NOT_EMPTY) can be returned. If this occurs, try the queue deletion again after a few seconds.

Opening the system-command input queue

Before you can open the system-command input queue, your application program must be connected to your queue manager. Use the MQI call **MQCONN** or **MQCONNX** to do this.

Then use the MQI call **MQOPEN** to open the system-command input queue. To use this call:

1. Set the *Options* parameter to MQOO_OUTPUT
2. Set the MQOD object descriptor fields as follows:

   ObjectType
   MQOT_Q (the object is a queue)

   ObjectName
   SYSTEM.COMMAND.INPUT

   ObjectQMgrName
   If you want to send your request messages to your local queue manager, leave this field blank. This means that your commands are processed locally.

   If you want your IBM MQ commands to be processed on a remote queue manager, put its name here. You must also have the correct queues and links set up, as described in Distributed queuing and clusters.

Opening a reply-to queue

To retrieve the replies from an IBM MQ command, you must open a reply-to queue. One way of doing this is to specify the model queue, SYSTEM.COMMAND.REPLY.MODEL in an **MQOPEN** call, to create a permanent dynamic queue as the reply-to queue. To use this call:

1. Set the *Options* parameter to MQOO_INPUT_SHARED
2. Set the MQOD object descriptor fields as follows:

   ObjectType
   MQOT_Q (the object is a queue)

   ObjectName
   The name of the reply-to queue. If the queue name you specify is the name of a model queue object, the queue manager creates a dynamic queue.

   ObjectQMgrName
   To receive replies on your local queue manager, leave this field blank.

   DynamicQName
   Specify the name of the dynamic queue to be created.

Using the command server

The command server is an IBM MQ component that works with the command processor component. You can send formatted messages to the command server which interprets the messages, runs the administration requests, and sends responses back to your administration application.

The command server reads request messages from the system-command input queue, verifies them, and passes the valid ones as commands to the command processor. The command processor processes the commands and puts any replies as reply messages on to the reply-to queue that you specify. The first reply message contains the user message CSQN205I. See “Interpreting the reply messages from the command server” on page 293 for more information. The command server also processes channel initiator and queue-sharing group commands, wherever they are issued from.

Identifying the queue manager that processes your commands

The queue manager that processes the commands you issue from an administration program is the queue manager that owns the system-command input queue that the message is put onto.

Starting the command server

Normally, the command server is started automatically when the queue manager is started. It becomes available as soon as the message CSQ9022I 'START QMGR' NORMAL COMPLETION is returned from the START QMGR command. The command server is stopped when all the connected tasks have been disconnected during the system termination phase.

You can control the command server yourself using the START CMDSERV and STOP CMDSERV commands. To prevent the command server starting automatically when IBM MQ is restarted, you can add a STOP CMDSERV command to your CSQINP1 or CSQINP2 initialization data sets. However, this is not recommended as it prevents any channel initiator or queue-sharing group commands being processed.

The STOP CMDSERV command stops the command server as soon as it has finished processing the current message, or immediately if no messages are being processed.

If the command server has been stopped by a STOP CMDSERV command in the program, no other commands from the program can be processed. To restart the command server, you must issue a START CMDSERV command from the z/OS console.

If you stop and restart the command server while the queue manager is running, all the messages that are on the system-command input queue when the command server stops are processed when the command server is restarted. However, if you stop and restart the queue manager after the command server is stopped, only the persistent messages on the system-command input queue are processed when the command server is restarted. All nonpersistent messages on the system-command input queue are lost.

Sending commands to the command server

For each command, you build a message containing the command, then put it onto the system-command input queue.

Building a message that includes IBM MQ commands

You can incorporate IBM MQ commands in an application program by building request messages that include the required commands. For each such command you:

1. Create a buffer containing a character string representing the command.
2. Issue an **MQPUT** call specifying the buffer name in the `buffer` parameter of the call.

The simplest way to do this in C is to define a buffer using 'char'. For example:

```
char message_buffer[ ] = "ALTER QLOCAL(SALES) PUT(ENABLED)";
```

When you build a command, use a null-terminated character string. Do not specify a command prefix string (CPF) at the start of a command defined in this way. This means that you do not have to alter your command scripts if you want to run them on another queue manager. However, you must take into account that a CPF is included in any response messages that are put onto the reply-to queue.

The command server folds all lowercase characters to uppercase unless they are inside quotation marks.

Commands can be any length up to a maximum 32 762 characters.

Putting messages on the system-command input queue

Use the **MQPUT** call to put request messages containing commands on the system-command input queue. In this call you specify the name of the reply-to queue that you have already opened.

To use the **MQPUT** call:

1. Set these **MQPUT** parameters:

   Hconn The connection handle returned by the **MQCONN** or **MQCONNX** call.

   Hobj The object handle returned by the **MQOPEN** call for the system-command input queue.

   BufferLength
   The length of the formatted command.

   Buffer The name of the buffer containing the command.
2. Set these MQMD fields:

   MsgType
   MQMT_REQUEST

   Format MQFMT_STRING or MQFMT_NONE

   If you are not using the same code page as the queue manager, set *CodedCharSetId* as appropriate and set MQFMT_STRING, so that the command server can convert the message. Do not set MQFMT_ADMIN, as that causes your command to be interpreted as PCF.

   ReplyToQ
   Name of your reply-to queue.

   ReplyToQMgr
   If you want replies sent to your local queue manager, leave this field blank. If you want your IBM MQ commands to be sent to a remote queue manager, put its name here. You must also have the correct queues and links set up, as described in Distributed queuing and clusters.
3. Set any other MQMD fields, as required. You should normally use nonpersistent messages for commands.
4. Set any *PutMsgOpts* options, as required.

   If you specify MQPMO_SYNCPOINT (the default), you must follow the **MQPUT** call with a syncpoint call.

Using MQPUT1 and the system-command input queue

If you want to put just one message on the system-command input queue, you can use the **MQPUT1** call. This call combines the functions of an **MQOPEN** , followed by an **MQPUT** of one message, followed by an **MQCLOSE** , all in one call. If you use this call, modify the parameters accordingly. See Putting one message on a queue using the MQPUT1 call for details.

Retrieving replies to your commands

The command server sends a response to a reply queue for each request message it receives. Any administration application must receive, and handle the reply messages.

When the command processor processes your commands, any reply messages are put onto the reply-to queue specified in the **MQPUT** call. The command server sends the reply messages with the same persistence as the command message it received.

Waiting for a reply

Use the **MQGET** call to retrieve a reply from your request message. One request message can produce several reply messages. For details, see "Interpreting the reply messages from the command server" on page 293.

You can specify a time interval that an **MQGET** call waits for a reply message to be generated. If you do not get a reply, use the checklist beginning in topic "If you do not receive a reply" on page 293.

To use the **MQGET** call:

1. Set these parameters:

   Hconn The connection handle returned by the **MQCONN** or **MQCONNX** call.

   Hobj The object handle returned by the **MQOPEN** call for the reply-to queue.

   Buffer The name of the area to receive the reply.

   BufferLength
   The length of the buffer to receive the reply. This must be a minimum of 80 bytes.
2. To ensure that you only get the responses from the command that you issued, you must specify the appropriate *MsgId* and *CorrelId* fields. These depend on the report options, MQMD_REPORT, you specified in the **MQPUT** call:

   MQRO_NONE
   Binary zero, '00...00' (24 nulls).

   MQRO_NEW_MSG_ID
   Binary zero, '00...00' (24 nulls).

   This is the default if none of these options has been specified.

   MQRO_PASS_MSG_ID
   The *MsgId* from the **MQPUT** .

   MQRO_NONE
   The *MsgId* from the **MQPUT** call.

   MQRO_COPY_MSG_ID_TO_CORREL_ID
   The *MsgId* from the **MQPUT** call.

   This is the default if none of these options has been specified.

   MQRO_PASS_CORREL_ID
   The *CorrelId* from the **MQPUT** call.

   For more details on report options, see Report options and message flags.
3. Set the following *GetMsgOpts* fields:

   Options
   MQGMO_WAIT

If you are not using the same code page as the queue manager, set MQGMO_CONVERT, and set *CodedCharSetId* as appropriate in the MQMD.

WaitInterval
For replies from the local queue manager, try 5 seconds. Coded in milliseconds, this becomes 5 000. For replies from a remote queue manager, and channel control and status commands, try 30 seconds. Coded in milliseconds, this becomes 30 000.

Discarded messages

If the command server finds that a request message is not valid, it discards this message and writes the message CSQN205I to the named reply-to queue. If there is no reply-to queue, the CSQN205I message is put onto the dead-letter queue. The return code in this message shows why the original request message was not valid:

00D5020F	It is not of type MQMT_REQUEST.
00D50210	It has zero length.
00D50212	It is longer than 32 762 bytes.
00D50211	It contains all blanks.
00D5483E	It needed converting, but *Format* was not MQFMT_STRING.
Other	See Command server codes

The command server reply message descriptor

For any reply message, the following MQMD message descriptor fields are set:

MsgType	MQMT_REPLY
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
Priority	As for the MQMD in the message you issued.
Persistence	As for the MQMD in the message you issued.
CorrelId	Depends on the **MQPUT** report options.
ReplyToQ	None.

The command server sets the *Options* field of the MQPMO structure to MQPMO_NO_SYNCPOINT. This means that you can retrieve the replies as they are created, rather than as a group at the next syncpoint.

Interpreting the reply messages from the command server

Each request message correctly processed by IBM MQ produces at least two reply messages. Each reply message contains a single IBM MQ user message.

The length of a reply depends on the command that was issued. The longest reply you can get is from a DISPLAY NAMELIST, and that can be up to 15 000 bytes in length.

The first user message, CSQN205I, always contains:

- A count of the replies (in decimal), which you can use as a counter in a loop to get the rest of the replies. The count includes this first message.
- The return code from the command preprocessor.
- A reason code, which is the return code from the command processor.

This message does not contain a CPF.

For example:

```
CSQN205I    COUNT=    4, RETURN=0000000C, REASON=00000008
```

The COUNT field is 8 bytes long and is right-justified. It always starts at position 18, that is, immediately after 'COUNT='. The RETURN field is 8 bytes long in character hexadecimal and is immediately after 'RETURN=' at position 35. The REASON field is 8 bytes long in character hexadecimal and is immediately after 'REASON=' at position 52.

If the RETURN= value is 00000000 and the REASON= value is 00000004, the set of reply messages is incomplete. After retrieving the replies indicated by the CSQN205I message, issue a further **MQGET** call to wait for a further set of replies. The first message in the next set of replies is again CSQN205I, indicating how many replies there are, and whether there are still more to come.

See the IBM MQ for z/OS messages, completion, and reason codes documentation for more details about the individual messages.

If you are using a non-English language feature, the text and layout of the replies are different from those shown here. However, the size and position of the count and return codes in message CSQN205I are the same.

If you do not receive a reply

There are a series of steps you can take if you do not receive a response to request to the command server.

If you do not receive a reply to your request message, work through this checklist:

- Is the command server running?
- Is the `WaitInterval` long enough?
- Are the system-command input and reply-to queues correctly defined?
- Were the **MQOPEN** calls to these queues successful?
- Are both the system-command input and reply-to queues enabled for **MQPUT** and **MQGET** calls?
- Have you considered increasing the MAXDEPTH and MAXMSGL attributes of your queues?
- Are you are using the `CorrelId` and `MsgId` fields correctly?
- Is the queue manager still running?
- Was the command built correctly?
- Are all your remote links defined and operating correctly?

- Were the **MQPUT** calls correctly defined?
- Has the reply-to queue been defined as a temporary dynamic queue instead of a permanent dynamic queue? (If the request message is persistent, you must use a permanent dynamic queue for the reply.)

When the command server generates replies but cannot write them to the reply-to queue that you specify, it writes them to the dead-letter queue.

Passing commands using MGCRE

With appropriate authorization, an application program can make requests to multiple queue managers using a z/OS service routine.

If you have the correct authorization, you can pass IBM MQ commands from your program to multiple queue managers by the MGCRE (SVC 34) z/OS service. The value of the CPF identifies the particular queue manager to which the command is directed. For information about CPFs, see User IDs for command security and command resource security and “Issuing queue manager commands” on page 266.

If you use MGCRE, you can use a Command and Response Token (CART) to get the direct responses to the command.

Examples of commands and their replies

Use this topic as a series of examples of commands to the command server and the responses from the command server.

Here are some examples of commands that could be built into IBM MQ messages, and the user messages that are the replies. Unless otherwise stated, each line of the reply is a separate message.
- Messages from a DEFINE command
- Messages from a DELETE command
- Messages from DISPLAY commands
- Messages from commands with CMDSCOPE
- Messages from commands that generate commands with CMDSCOPE

Messages from a DEFINE command

The following command:

```
DEFINE QLOCAL(Q1)
```

produces these messages:

```
CSQN205I    COUNT=    2, RETURN=00000000, REASON=00000000
CSQ9022I +CSQ1 CSQMMSGP ' DEFINE QLOCAL' NORMAL COMPLETION
```

These reply messages are produced on normal completion.

Messages from a DELETE command

The following command:

```
DELETE QLOCAL(Q2)
```

produces these messages:

```
CSQN205I    COUNT=    4, RETURN=0000000C, REASON=00000008
CSQM125I +CSQ1 CSQMUQLC QLOCAL (Q2) QSGDISP(QMGR) WAS NOT FOUND
CSQM090E +CSQ1 CSQMUQLC FAILURE REASON CODE X'00D44002'
CSQ9023E +CSQ1 CSQMUQLC ' DELETE QLOCAL' ABNORMAL COMPLETION
```

These messages indicate that a local queue called Q2 does not exist.

Messages from DISPLAY commands

The following examples show the replies from some DISPLAY commands.

Finding out the name of the dead-letter queue

If you want to find out the name of the dead-letter queue for a queue manager, issue this command from an application program:

```
DISPLAY QMGR DEADQ
```

The following three user messages are returned, from which you can extract the required name:

```
CSQN205I    COUNT=    3, RETURN=00000000, REASON=00000000
CSQM409I +CSQ1 QMNAME(CSQ1) DEADQ(SYSTEM.DEAD.QUEUE          )
CSQ9022I +CSQ1 CSQMDRTS ' DISPLAY QMGR' NORMAL COMPLETION
```

Messages from the DISPLAY QUEUE command

The following examples show how the results from a command depend on the attributes specified in that command.

Example 1

You define a local queue using the command:

```
DEFINE QLOCAL(Q1) DESCR('A sample queue') GET(ENABLED) SHARE
```

If you issue the following command from an application program:

```
DISPLAY QUEUE(Q1) SHARE GET DESCR
```

these three user messages are returned:

```
CSQN205I    COUNT=    3, RETURN=00000000, REASON=00000000
CSQM401I +CSQ1 QUEUE(Q1                       ) TYPE(
QLOCAL ) QSGDISP(QMGR   )
DESCR(A sample queue
) SHARE  GET(ENABLED  )
CSQ9022I +CSQ1 CSQMDMSG ' DISPLAY QUEUE' NORMAL COMPLETION
```

Note: The second message, CSQM401I, is shown here occupying four lines.

Example 2

Two queues have names beginning with the letter A:

- A1 is a local queue with its PUT attribute set to DISABLED.
- A2 is a remote queue with its PUT attribute set to ENABLED.

If you issue the following command from an application program:

```
DISPLAY QUEUE(A*) PUT
```

these four user messages are returned:

```
CSQN205I    COUNT=    4, RETURN=00000000, REASON=00000000
CSQM401I +CSQ1 QUEUE(A1                       ) TYPE(
QLOCAL ) QSGDISP(QMGR   )
PUT(DISABLED )
CSQM406I +CSQ1 QUEUE(A2                       ) TYPE(
QREMOTE ) PUT(ENABLED  )
CSQ9022I +CSQ1 CSQMDMSG ' DISPLAY QUEUE' NORMAL COMPLETION
```

Note: The second and third messages, CSQM401I and CSQM406I, are shown here occupying three and two lines.

Messages from the DISPLAY NAMELIST command

You define a namelist using the command:

```
DEFINE NAMELIST(N1) NAMES(Q1,SAMPLE_QUEUE)
```

If you issue the following command from an application program:

```
DISPLAY NAMELIST(N1) NAMES NAMCOUNT
```

the following three user messages are returned:

```
CSQN205I    COUNT=    3, RETURN=00000000, REASON=00000000
CSQM407I +CSQ1 NAMELIST(N1                          ) QS
GDISP(QMGR   ) NAMCOUNT(     2) NAMES(Q1
,SAMPLE_QUEUE                  )
CSQ9022I +CSQ1 CSQMDMSG ' DISPLAY NAMELIST' NORMAL COMPLETION
```

Note: The second message, CSQM407I, is shown here occupying three lines.

Messages from commands with CMDSCOPE

The following examples show the replies from commands that have been entered with the CMDSCOPE attribute.

Messages from the ALTER PROCESS command

The following command:

```
ALT PRO(V4) CMDSCOPE(*)
```

produces the following messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'ALT PRO' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    5, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'ALT PRO' command responses from MQ26
CSQM125I !MQ26 CSQMMSGP PROCESS(V4) QSGDISP(QMGR) WAS NOT FOUND
CSQM090E !MQ26 CSQMMSGP FAILURE REASON CODE X'00D44002'
CSQ9023E !MQ26 CSQMMSGP ' ALT PRO' ABNORMAL COMPLETION
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'ALT PRO' command responses from MQ25
CSQ9022I !MQ25 CSQMMSGP ' ALT PRO' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=0000000C, REASON=00000008
CSQN123E !MQ25 'ALT PRO' command for CMDSCOPE(*) abnormal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). The command was successful on MQ25 but the process definition did not exist on MQ26, so the command failed on that queue manager.

Messages from the DISPLAY PROCESS command

The following command:

```
DIS PRO(V*) CMDSCOPE(*)
```

produces the following messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'DIS PRO' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    5, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS PRO' command responses from MQ26
CSQM408I !MQ26 PROCESS(V2) QSGDISP(COPY)
CSQM408I !MQ26 PROCESS(V3) QSGDISP(QMGR)
CSQ9022I !MQ26 CSQMDRTS ' DIS PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    7, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS PRO' command responses from MQ25
CSQM408I !MQ25 PROCESS(V2) QSGDISP(COPY)
CSQM408I !MQ25 PROCESS(V2) QSGDISP(GROUP)
CSQM408I !MQ25 PROCESS(V3) QSGDISP(QMGR)
CSQM408I !MQ25 PROCESS(V4) QSGDISP(QMGR)
CSQ9022I !MQ25 CSQMDRTS ' DIS PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'DIS PRO' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). Information is displayed about all the processes on each queue manager with names starting with the letter V.

Messages from the DISPLAY CHSTATUS command

The following command:

```
DIS CHS(VT) CMDSCOPE(*)
```

produces the following messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'DIS CHS' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    4, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS CHS' command responses from MQ25
CSQM422I !MQ25 CHSTATUS(VT) CHLDISP(PRIVATE) CONNAME( ) CURRENT STATUS(STOPPED)
CSQ9022I !MQ25 CSQXDRTS ' DIS CHS' NORMAL COMPLETION
CSQN205I  COUNT=    4, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DIS CHS' command responses from MQ26
CSQM422I !MQ26 CHSTATUS(VT) CHLDISP(PRIVATE) CONNAME( ) CURRENT STATUS(STOPPED)
CSQ9022I !MQ26 CSQXDRTS ' DIS CHS' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'DIS CHS' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). Information is displayed about channel status on each queue manager.

Messages from the STOP CHANNEL command

The following command:

```
STOP CHL(VT) CMDSCOPE(*)
```

produces these messages:

```
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000004
CSQN137I !MQ25 'STOP CHL' command accepted for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ25
CSQM134I !MQ25 CSQMTCHL STOP CHL(VT) COMMAND ACCEPTED
SQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ26
CSQM134I !MQ26 CSQMTCHL STOP CHL(VT) COMMAND ACCEPTED
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ26
CSQ9022I !MQ26 CSQXCRPS ' STOP CHL' NORMAL COMPLETION
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'STOP CHL' command responses from MQ25
CSQ9022I !MQ25 CSQXCRPS ' STOP CHL' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'STOP CHL' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25 and sent to two queue managers (MQ25 and MQ26). Channel VT was stopped on each queue manager.

Messages from commands that generate commands with CMDSCOPE

The following command:

```
DEF PRO(V2) QSGDISP(GROUP)
```

produces these messages:

```
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQM122I !MQ25 CSQMMSGP ' DEF PRO' COMPLETED FOR QSGDISP(GROUP)
CSQN138I !MQ25 'DEFINE PRO' command generated for CMDSCOPE(*), sent to 2
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DEFINE PRO' command responses from MQ25
CSQ9022I !MQ25 CSQMMSGP ' DEFINE PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    3, RETURN=00000000, REASON=00000004
CSQN121I !MQ25 'DEFINE PRO' command responses from MQ26
CSQ9022I !MQ26 CSQMMSGP ' DEFINE PROCESS' NORMAL COMPLETION
CSQN205I  COUNT=    2, RETURN=00000000, REASON=00000000
CSQN122I !MQ25 'DEFINE PRO' command for CMDSCOPE(*) normal completion
```

These messages tell you that the command was entered on queue manager MQ25. When the object was created on the shared repository, another command was generated and sent to all the active queue managers in the queue-sharing group (MQ25 and MQ26).

Managing IBM MQ resources on z/OS

z/OS

Use the links in this topic to find out how to manage the resources used by IBM MQ for z/OS, for example, managing log files, data sets, page sets, buffer pools, and coupling facility structures.

Use the following links for details of the different administrative tasks you might have to complete while using IBM MQ for z/OS:

- "Managing the logs"
- "Managing the bootstrap data set (BSDS)" on page 309
- "Managing page sets" on page 317
- "How to back up and recover page sets" on page 323
- "How to back up and restore queues using CSQUTIL" on page 327
- "Managing buffer pools" on page 327
- "Managing queue-sharing groups and shared queues" on page 328

Related concepts:

"Administering IBM MQ for z/OS" on page 255
Administering queue managers and associated resources includes the tasks that you perform frequently to activate and manage those resources. Choose the method you prefer to administer your queue managers and associated resources.

"Issuing commands to IBM MQ for z/OS" on page 256
You can use IBM MQ script commands (MQSC) in batch or interactive mode to control a queue manager.

"Recovery and restart" on page 337
Use this topic to understand the recovery and restart mechanisms used by IBM MQ.

Related reference:

"The IBM MQ for z/OS utilities" on page 264
IBM MQ for z/OS provides a set of utility programs that you can use to help with system administration.

Related information:

IBM MQ for z/OS concepts

Planning your IBM MQ environment on z/OS

Configuring z/OS

Programmable command formats reference

MQSC reference

Using the IBM MQ for z/OS utilities

Managing the logs

Use this topic to understand how to manage your IBM MQ log files, including the log archiving process, using log record compression, log record recovery, and printing log records.

This topic describes the tasks involved in managing the IBM MQ logs. It contains these sections:

Archiving logs with the ARCHIVE LOG command:

An authorized operator can archive the current IBM MQ active log data sets whenever required using the ARCHIVE LOG command.

When you issue the ARCHIVE LOG command, IBM MQ truncates the current active log data sets, then runs an asynchronous offload process, and updates the BSDS with a record of the offload process.

The ARCHIVE LOG command has a MODE(QUIESCE) option. With this option, IBM MQ jobs and users are quiesced after a commit point, and the resulting point of consistency is captured in the current active log before it is offloaded.

Consider using the MODE(QUIESCE) option when planning a backup strategy for off site recovery. It creates a system-wide point of consistency, which minimizes the number of data inconsistencies when the archive log is used with the most current backup page set copy during recovery. For example:

```
ARCHIVE LOG MODE(QUIESCE)
```

If you issue the ARCHIVE LOG command without specifying a TIME parameter, the quiesce time period defaults to the value of the QUIESCE parameter of the CSQ6ARVP macro. If the time required for the ARCHIVE LOG MODE(QUIESCE) to complete is less than the time specified, the command completes successfully; otherwise, the command fails when the time period expires. You can specify the time period explicitly by using the TIME option, for example:

```
ARCHIVE LOG MODE(QUIESCE) TIME(60)
```

This command specifies a quiesce period of up to 60 seconds before ARCHIVE LOG processing occurs.

Attention: Using the TIME option when time is critical can significantly disrupt IBM MQ availability for all jobs and users that use IBM MQ resources.

By default, the command is processed asynchronously from the time you submit the command. (To process the command synchronously with other IBM MQ commands use the WAIT(YES) option with QUIESCE, but be aware that the z/OS console is locked from IBM MQ command input for the entire QUIESCE period.)

During the quiesce period:

- Jobs and users on the queue manager are allowed to go through commit processing, but are suspended if they try to update any IBM MQ resource after the commit.
- Jobs and users that only read data can be affected, since they might be waiting for locks held by jobs or users that were suspended.
- New tasks can start, but they cannot update data.

The output from the DISPLAY LOG command uses the message CSQV400I to indicate that a quiesce is in effect. For example:

```
CSQJ322I +CSQ1 DISPLAY LOG report ...
Parameter   Initial value         SET value
----------- --------------------- ---------------------
INBUFF      60
OUTBUFF     4000
MAXRTU      2
MAXARCH     2
TWOACTV     YES
TWOARCH     YES
TWOBSDS     YES
OFFLOAD     YES
WRTHRSH     20
DEALLCT     0
End of LOG report
CSQJ370I +CSQ1 LOG status report ...
Copy %Full DSName
1      68  VICY.CSQ1.LOGCOPY1.DS01
2      68  VICY.CSQ1.LOGCOPY2.DS01
Restarted at 2014-04-15 09:49:30 using RBA=000000000891B000
Latest RBA=000000000891CCF8
Offload task is AVAILABLE
Full logs to offload - 0 of 4
CSQV400I +CSQ1 ARCHIVE LOG QUIESCE CURRENTLY ACTIVE
CSQ9022I +CSQ1 CSQJC001 ' DISPLAY LOG' NORMAL COMPLETION
```

When all updates are quiesced, the quiesce history record in the BSDS is updated with the date and time that the active log data sets were truncated, and with the last-written RBA in the current active log data sets. IBM MQ truncates the current active log data sets, switches to the next available active log data sets, and issues message CSQJ311I stating that the offload process started.

If updates cannot be quiesced before the quiesce period expires, IBM MQ issues message CSQJ317I, and ARCHIVE LOG processing terminates. The current active log data sets are not truncated, nor switched to the next available log data sets, and the offload process is not started.

Whether the quiesce was successful or not, all suspended users and jobs are then resumed, and IBM MQ issues message CSQJ312I, stating that the quiesce is ended and update activity is resumed.

If ARCHIVE LOG is issued when the current active log is the last available active log data set, the command is not processed, and IBM MQ issues the following message:

```
CSQJ319I - csect-name CURRENT ACTIVE LOG DATA SET IS THE LAST
AVAILABLE ACTIVE LOG DATA SET. ARCHIVE LOG PROCESSING
WILL BE TERMINATED
```

If ARCHIVE LOG is issued when another ARCHIVE LOG command is already in progress, the new command is not processed, and IBM MQ issues the following message:

```
CSQJ318I - ARCHIVE LOG COMMAND ALREADY IN PROGRESS
```

For information about the messages issued during archiving, see Messages for IBM MQ for z/OS.

Restarting the log archive process after a failure

If there is a problem during the log archive process (for example, a problem with allocation or tape mounts), the archiving of the active log might be suspended. You can cancel the archive process and restart it by using the ARCHIVE LOG CANCEL OFFLOAD command. This command cancels any offload processing currently in progress, and restarts the archive process. It starts with the oldest log data set that has not been archived, and proceeds through all active log data sets that need offloading. Any log archive operations that have been suspended are restarted.

Use this command only if you are sure that the current log archive task is no longer functioning, or if you want to restart a previous attempt that failed. This is because the command might cause an abnormal termination of the offload task, which might result in a dump.

Controlling archiving and logging:

You can control compression, printing, archiving, recovery and logging with using the CSQ6LOGP, CSQ6ARVP, and CSQ6SYSP macros.

Many aspects of archiving and logging are controlled by parameters set using the CSQ6LOGP, CSQ6ARVP and CSQ6SYSP macros of the system parameter module when the queue manager is customized. See Task 17: Tailor your system parameter module for details of these macros.

Some of these parameters can be changed while a queue manager is running using the IBM MQ MQSC SET LOG, SET SYSTEM and SET ARCHIVE commands. They are shown in Table 14:

Table 14. Archiving and logging parameters that can be changed while a queue manager is running

SET command	Parameters
LOG	WRTHRSH, MAXARCH, DEALLCT, MAXRTU, COMPLOG
ARCHIVE	All
SYSTEM	LOGLOAD

You can display the settings of all the parameters using the MQSC DISPLAY LOG, DISPLAY ARCHIVE and DISPLAY SYSTEM commands. These commands also show status information about archiving and logging.

Controlling log compression

You can enable and disable the compression of log records using either

- The SET and DISPLAY LOG commands in MQSC; see The MQSC commands
- Invoking PCF interface. See “Introduction to Programmable Command Formats” on page 6
- Using the CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

Printing log records

You can extract and print log records using the CSQ1LOGP utility. For instructions, see The log print utility.

Recovering logs

Normally, you do not need to back up and restore the IBM MQ logs, especially if you are using dual logging. However, in rare circumstances, such as an I/O error on a log, you might need to recover the logs. Use Access Method Services to delete and redefine the data set, and then copy the corresponding dual log into it. For more information see Using Access Method Services.

Discarding archive log data sets:

You can discard your archive log data sets and choose to discard the logs automatically or manually.

You must keep enough log data to be able to perform unit of work recovery, page set media recovery if a page set is lost, or CF structure media recovery if a CF structure is lost. Do not discard archive log data sets that might be required for recovery; if you discard these archive log data sets you might not be able to perform required recovery operations.

If you have confirmed that your archive log data sets can be discarded, you can do this in either of the following ways:
- Automatic archive log data set deletion
- Manually deleting archive log data sets

Automatic archive log data set deletion

You can use a DASD or tape management system to delete archive log data sets automatically. The retention period for IBM MQ archive log data sets is specified by the retention period field ARCRETN in the CSQ6ARVP installation macro (see the Using CSQ6ARVP for more information).

The default for the retention period specifies that archive logs are to be kept for 9999 days (the maximum). **You can change the retention period but you must ensure that you can accommodate the number of backup cycles that you have planned for**.

IBM MQ uses the retention period value as the value for the JCL parameter RETPD when archive log data sets are created.

The retention period set by the MVS™/DFP storage management subsystem (SMS) can be overridden by this IBM MQ parameter. Typically, the retention period is set to the smaller value specified by either IBM MQ or SMS. The storage administrator and IBM MQ administrator must agree on a retention period value that is appropriate for IBM MQ.

Note: IBM MQ does not have an automated method to delete information about archive log data sets from the BSDS, because some tape management systems provide external manual overrides of retention periods. Therefore, information about an archive log data set can still be in the BSDS long after the data set retention period has expired and the data set has been scratched by the tape management system. Conversely, the maximum number of archive log data sets might have been exceeded and the data from the BSDS might have been dropped before the data set has reached its expiration date.

If archive log data sets are deleted automatically, remember that the operation does not update the list of archive logs in the BSDS. You can update the BSDS with the change log inventory utility, as described in “Changing the BSDS” on page 311. The update is not essential. Recording old archive logs wastes space in the BSDS, but does no other harm.

Manually deleting archive log data sets

You must keep all the log records as far back as the lowest RBA identified in messages CSQI024I and CSQI025I. This RBA is obtained using the DISPLAY USAGE command that you issued when creating a point of recovery using Method 1: Full backup.

Read Creating a point of recovery for non-shared resources before discarding any logs.

Locate and discard archive log data sets

Having established the minimum log RBA required for recovery, you can find archive log data sets that contain only earlier log records by performing the following procedure:

1. Use the print log map utility to print the contents of the BSDS. For an example of the output, see The print log map utility.
2. Find the sections of the output titled "ARCHIVE LOG COPY n DATA SETS". If you use dual logging, there are two sections. The columns labeled STARTRBA and ENDRBA show the range of RBAs contained in each volume. Find the volumes with ranges that include the minimum RBA you found with messages CSQI024I and CSQI025I. These are the earliest volumes you need to keep. If you are using dual-logging, there are two such volumes.

   If no volumes have an appropriate range, one of the following cases applies:

   - The minimum RBA has not yet been archived, and you can discard all archive log volumes.
   - The list of archive log volumes in the BSDS wrapped around when the number of volumes exceeded the number allowed by the MAXARCH parameter of the CSQ6LOGP macro. If the BSDS does not register an archive log volume, that volume cannot be used for recovery. Therefore, consider adding information about existing volumes to the BSDS. For instructions, see “Changes for archive logs” on page 313.

     Also consider increasing the value of MAXARCH. For information, see the Using CSQ6LOGP.
3. Delete any archive log data set or volume with an ENDRBA value that is less than the STARTRBA value of the earliest volume you want to keep. If you are using dual logging, delete both such copies.

   Because BSDS entries wrap around, the first few entries in the BSDS archive log section might be more recent than the entries at the bottom. Look at the combination of date and time and compare their ages. Do not assume that you can discard all entries *above* the entry for the archive log containing the minimum LOGRBA.

   Delete the data sets. If the archives are on tape, erase the tapes. If they are on DASD, run a z/OS utility to delete each data set. Then, if you want the BSDS to list only existing archive volumes, use the change log inventory utility (CSQJU003) to delete entries for the discarded volumes. See “Changes for archive logs” on page 313 for an example.

The effect of log shunting:

Long running transactions can cause unit of work log records which span log data sets. IBM MQ handles this scenario by using log shunting, a technique which moves the log records to optimize the quantity of log data retained, and queue manager restart time.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This is known as *log shunting*. It is described more fully in Log files.

The queue manager uses these shunted log records instead of the originals after a failure, to ensure unit of work integrity. There are two benefits to this:

- the quantity of log data which must be retained for unit of work coordination is reduced
- less log data must be traversed at queue manager restart time, so the queue manager is restarted more quickly

Shunted log records do not contain sufficient information for media recovery operations.

Data held in the log is used for two distinct purposes; media recovery and unit of work coordination. If a media failure occurs which affects either a CF structure or page set, the queue manager can recover the media to the point of failure by restoring a prior copy and updating this using data contained in the log. Persistent activity performed in a unit of work is recorded on the log so that in the event of a failure, it can either be backed out or locks can be recovered on changed resources. The quantity of log data you need to retain to enable queue manager recovery is affected by these two elements.

For media recovery, you must retain sufficient log data to be able to perform media recovery from at least the most recent media copy and to be able to back out. (Your site may stipulate the ability to recover from older backups.) For unit of work integrity, you must retain the log data for your oldest in flight or indoubt units of work.

To assist you with managing the system, the queue manager detects old units of work at each log archive and reports them in messages CSQJ160 and CSQJ161. An internal task reads unit of work log information for these old units of work and rewrites it in a more succinct form to the current position in the log. Message CSQR026 indicates when this has happened. The MQSC command DISPLAY USAGE TYPE(DATASET) can also help you to manage the retention of log data. The command reports 3 pieces of recovery information:

1. how much of the log must be retained for unit of work recovery
2. how much of the log must be retained for media recovery of page sets
3. for a queue manager in a queue-sharing group, how much of the log must be retained for media recovery of CF structures

For each of these, an attempt is made to map the oldest log data required into a data set. As new units of work start and stop, we would expect (1) above to move to a more recent position in the log. If it is not moving, the long running UOW messages warn you that there is an issue. (2) relates to page set media recovery if the queue manager were to be shut down now and restarted. It does not know about when you last backed up your page sets, or which backup you might have to use if there was a page set failure. It normally moves to a more recent position in the log during checkpoint processing as changes held in the buffer pools are written to the page sets. In (3), the queue manager does know about CF structure backups taken either on this queue manager or on other queue managers in the queue sharing group. However, CF structure recovery requires a merge of log data from all queue managers in the queue-sharing group which have interacted with the CF structure since the last backup. This means that the log data is identified by a log record sequence number, (or LRSN), which is timestamp based and so applicable across the entire queue-sharing group rather than an RBA which would be different on different queue managers in the queue-sharing group. It normally moves to a more recent position in the log as BACKUP CFSTRUCT commands are performed on either this or other queue managers in the queue-sharing group.

Resetting the queue manager's log:

Use this topic to understand how to reset the queue manager's log.

You must not allow the queue manager log RBA to wrap around from the end of the log RBA range to 0, as this leads to a queue manager outage and all persistent data will become unrecoverable. The end of the log RBA is either a value of FFFFFFFFFFFF (if 6-byte RBAs as in use), or FFFFFFFFFFFFFFFF (if 8-byte RBAs are in use).

The queue manager issues messages CSQI045I, CSQI046E, CSQI047E, and CSQJ032E to indicate that the used log range is significant and that you should plan to take action to avoid an unplanned outage.

The queue manager terminates with reason code 00D10257 when the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use).

If 6-byte log RBAs are in use, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the process described in Implementing the larger log Relative Byte Address. Converting a queue manager to use 8-byte log RBAs requires a shorter outage than resetting the log, and increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6-byte or 8-byte log RBAs are in use.

The procedure to follow to reset the queue manager's log is as follows:

1. Resolve any unresolved units of work. The number of unresolved units of work is displayed at queue manager startup in message CSQR005I as the INDOUBT count. At each checkpoint, and at queue manager shutdown, the queue manager automatically issues the command

   `DISPLAY CONN(*) TYPE(CONN) ALL WHERE(UOWSTATE EQ UNRESOLVED)` to provide information about unresolved units of work.

   See How in-doubt units of recovery are resolved for information on resolving units of recovery. The ultimate recourse is to use the `RESOLVE INDOUBT` MQSC command to manually resolve indoubt units of recovery.
2. Shutdown the queue manager cleanly.

   You can use either `STOP QMGR` or `STOP QMGR MODE(FORCE)` as both these commands flush any changed pages from bufferpools to the pagesets.
3. If a queue manager is part of a queue sharing group, take CFSTRUCT backups on other queue managers for all structures in the queue sharing group. This ensures that the most recent backups are not in this queue manager's log, and that this queue manager's log is not required for CFSTRUCT recovery.
4. Define new logs and BSDS using CSQJU003 (see The change log inventory utility for more information on using the change log inventory utility).
5. Run `CSQUTIL RESETPAGE` against all the page sets for this queue manager (see Copying a page and resetting the log for more information on using this function). Note that page set RBAs can be reset independently, so multiple concurrent jobs (for example, one per page set) can be submitted to reduce the elapsed time for this step.
6. Restart the queue manager

Related concepts:

../com.ibm.mq.adm.doc/q119810_.dita
Previous releases of IBM MQ for z/OS used a 6-byte log RBA to identify the location of data within the log. In IBM MQ Version 8.0, the log RBA can be 8 bytes long, increasing the period of time before you have to reset the log.

Implementing the larger log Relative Byte Address:

Previous releases of IBM MQ for z/OS used a 6-byte log RBA to identify the location of data within the log. In IBM MQ Version 8.0, the log RBA can be 8 bytes long, increasing the period of time before you have to reset the log.

This new feature needs to be explicitly enabled. See Planning to increase the maximum addressable log range for considerations when planning to enable 8 byte log RBA.

Perform these instructions, in the order shown, to enable 8 byte log RBA on a single IBM MQ for z/OS queue manager:

1. Enable IBM MQ Version 8.0 new functions using OPMODE.

   For queue managers in a queue-sharing group, you do not need to take a total queue-sharing group outage. You can stop each queue manager in turn, enable it for OPMODE=(NEWFUNC,800) and restart it.

   Once all queue managers in the queue sharing group are running with OPMODE(NEWFUNC,800), perform the following steps for each queue manager in the queue-sharing group until all queue managers are running with the new BSDS.

2. Allocate new BSDS data sets with similar attributes to the current BSDS. You can tailor sample CSQ4BSDS and delete any irrelevant statement, or you can use your existing JCL, but change the BSDS name to something like `++HLQ++.NEW.BSDS01`.

   Notes:

   a. Check the attributes of your new BSDS. The only attribute that might change is the size of the BSDS.
   b. The new BSDS contains more data that the current BSDS, therefore, you must ensure that the new data sets are allocated with sufficient available space. See Planning your logging environment, and the associated topics, for the recommended values when defining a new BSDS.
3. Shut down the queue manager cleanly.
4. Run the BSDS conversion utility (CSQJUCNV) to convert the existing BSDS to the new BSDS data sets. This usually takes a few seconds to run.

   Your existing BSDS will not be changed during this process, and you can use that for the initialization of the queue manager in the case of an unsuccessful conversion.
5. Rename the current BSDS to become the old BSDS, and the new BSDS to become the current BSDS, so that the new data sets are used when you next restart the queue manager. You can use the DFSMS Access Method Services ALTER command, for example:
   ```
   ALTER '++HLQ++.BSDS01' NEWNAME('++HLQ++.OLD.BSDS01')
   ALTER '++HLQ++.NEW.BSDS01' NEWNAME('++HLQ++.BSDS01')
   ```
   Ensure that you also issue commands to rename both the data and index portions of the VSAM cluster.
6. Restart the queue manager. It should start in the same amount of time as it would have done when using 6 byte log RBA.

   If the queue manager does not restart successfully due to a failure to access the converted BSDS, attempt to identify the cause of the failure, resolve the problem and retry the operation. If required, contact your IBM support center for assistance.

   If necessary, the change can be backed out at this point by:
   a. Renaming the current BSDS to become the new BSDS.
   b. Renaming the old BSDS to become the current BSDS.
   c. Restarting the queue manager.

   Once the queue manager has been successfully restarted with the converted BSDS, do not attempt to start the queue manager using the old BSDS.
7. Message CSQJ034I is issued during queue manager initialization to indicate the end of the log RBA for the queue manager as configured. Confirm that the end of the log RBA range displayed is FFFFFFFFFFFFFFFF. This indicates that 8 byte log RBA is in use.

Note: In order to enable an 8 byte log RBA on a new IBM MQ Version 8.0 queue manager, before it is first started, you must first create an empty version 1 format BSDS and use that as input to the BSDS conversion utility to produce a version 2 format BSDS. See Create the bootstrap and log datasets for information on how you carry out this process.

Related information:

Planning to increase the maximum addressable log range

Larger log Relative Byte Address

The BSDS conversion utility (CSQJUCNV)

Managing the bootstrap data set (BSDS)

The bootstrap data set (BSDS) is used to reference log data sets, and log records. Use this topic to understand how you can examine, change, and recover the BSDS.

For more information, see The bootstrap data set.

This topic describes the tasks involved in managing the bootstrap data set. It contains these sections:
- "Finding out what the BSDS contains"
- "Changing the BSDS" on page 311
- "Recovering the BSDS" on page 315

Finding out what the BSDS contains:

You can use the print log map utility (CSQJU004) to examine the contents of the BSDS.

The print log map utility (CSQJU004) is a batch utility that lists the information stored in the BSDS. For instructions on running it, see The print log map utility.

The BSDS contains:
- Time stamps
- Active log data set status

Time stamps in the BSDS

The output of the print log map utility shows the time stamps, which are used to record the date and time of various system events, that are stored in the BSDS.

The following time stamps are included in the header section of the report:

SYSTEM TIMESTAMP

Reflects the date and time the BSDS was last updated. The BSDS time stamp can be updated when:
- The queue manager starts.
- The write threshold is reached during log write activities. Depending on the number of output buffers you have specified and the system activity rate, the BSDS might be updated several times a second, or might not be updated for several seconds, minutes, or even hours. For details of the write threshold, see the WRTHRSH parameter of the CSQ6LOGP macro in Using CSQ6LOGP.
- IBM MQ drops into a single BSDS mode from its normal dual BSDS mode due to an error. This can occur when a request to get, insert, point to, update, or delete a BSDS record is unsuccessful. When this error occurs, IBM MQ updates the time stamp in the remaining BSDS to force a time stamp mismatch with the disabled BSDS.

UTILITY TIMESTAMP

The date and time the contents of the BSDS were altered by the change log inventory utility (CSQJU003).

The following time stamps are included in the active and archive log data sets portion of the report:

Active log date
The date the active log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done.

Active log time
The time the active log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done.

Archive log date
The date the archive log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done or the archive itself was done.

Archive log time
The time the archive log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done or the archive itself was done.

Active log data set status

The BSDS records the status of an active log data set as one of the following:

NEW The data set has been defined but never used by IBM MQ, or the log was truncated to a point before the data set was first used. In either case, the data set starting and ending RBA values are reset to zero.

REUSABLE
Either the data set has been defined but never used by IBM MQ, or the data set has been offloaded. In the print log map output, the start RBA value for the last REUSABLE data set is equal to the start RBA value of the last archive log data set.

NOT REUSABLE
The data set contains records that have not been offloaded.

STOPPED
The offload processor encountered an error while reading a record, and that record could not be obtained from the other copy of the active log.

TRUNCATED
Either:

- An I/O error occurred, and IBM MQ has stopped writing to this data set. The active log data set is offloaded, beginning with the starting RBA and continuing up to the last valid record segment in the truncated active log data set. The RBA of the last valid record segment is lower than the ending RBA of the active log data set. Logging is switched to the next available active log data set, and continues uninterrupted.

or

- An ARCHIVE LOG function has been called, which has truncated the active log.

The status appears in the output from the print log map utility.

Changing the BSDS:

You do not have to take special steps to keep the BSDS updated with records of logging events because IBM MQ does that automatically.

However, you might want to change the BSDS if you do any of the following:
- Add more active log data sets.
- Copy active log data sets to newly allocated data sets, for example, when providing larger active log allocations.
- Move log data sets to other devices.
- Recover a damaged BSDS.
- Discard outdated archive log data sets.

You can change the BSDS by running the change log inventory utility (CSQJU003). Only run this utility when the queue manager is inactive, or you might get inconsistent results. The action of the utility is controlled by statements in the SYSIN data set. This section shows several examples. For complete instructions, see The change log inventory utility.

You can copy an active log data set only when the queue manager is inactive because IBM MQ allocates the active log data sets as exclusive (DISP=OLD) at queue manager startup.

Changes for active logs:

Use this topic to understand how you can change the active logs using the BSDS.

You can add to, delete from, and record entries in the BSDS for active logs using the change log utility. Examples only are shown here; replace the data set names shown with the ones you want to use. For more details of the utility, see The change log inventory utility.

See these sections for more information:
- Adding record entries to the BSDS
- Deleting information about the active log data set from the BSDS
- Recording information about the log data set in the BSDS
- Increasing the size of the active log
- The use of CSQJUFMT

Adding record entries to the BSDS

If an active log has been flagged as "stopped", it is not reused for logging; however, it continues to be used for reading. Use the access method services to define new active log data sets, then use the change log inventory utility to register the new data sets in the BSDS. For example, use:

```
NEWLOG DSNAME=MQM111.LOGCOPY1.DS10,COPY1
NEWLOG DSNAME=MQM111.LOGCOPY2.DS10,COPY2
```

If you are copying the contents of an old active log data set to the new one, you can also give the RBA range and the starting and ending time stamps on the NEWLOG function.

Deleting information about the active log data set from the BSDS

To delete information about an active log data set from the BSDS, you could use:

```
DELETE DSNAME=MQM111.LOGCOPY1.DS99
DELETE DSNAME=MQM111.LOGCOPY2.DS99
```

Recording information about the log data set in the BSDS

To record information about an existing active log data set in the BSDS, use:

```
NEWLOG DSNAME=MQM111.LOGCOPY1.DS10,COPY2,STARTIME=19930212205198,
ENDTIME=19930412205200,STARTRBA=6400,ENDRBA=94FF
```

You might need to insert a record containing this type of information in the BSDS because:

- The entry for the data set has been deleted, but is needed again.
- You are copying the contents of one active log data set to another data set.
- You are recovering the BSDS from a backup copy.

Increasing the size of the active log

There are two methods of achieving this process.

1. When the queue manager is active:
   a. Define new larger log data sets using JCL.
   b. Add the new log data sets to the active queue manager using the MQSC DEFINE LOG command.
   c. Use the MQSC ARCHIVE LOG command to move the current active log, to be a new larger log.
   d. Wait for the archive of the smaller active log dataset to complete.
   e. Shutdown the queue manager, using the CSQJU003 utility to remove the old small active logs.
   f. Restart the queue manager.
2. When the queue manager is inactive:
   a. Stop the queue manager. This step is required because IBM MQ allocates all active log data sets for its exclusive use when it is active.
   b. Use Access Method Services ALTER with the NEWNAME option to rename your active log data sets.
   c. Use Access Method Services DEFINE to define larger active log data sets.

      By reusing the old data set names, you do not have to run the change log inventory utility to establish new names in the BSDSs. The old data set names and the correct RBA ranges are already in the BSDSs.
   d. Use Access Method Services REPRO to copy the old (renamed) data sets into their appropriate new data sets.

      Note: This step can take a long time, so your enterprise could be out of action for this period.
   e. Start the queue manager.

If all your log data sets are the same size, your system will be operationally more consistent and efficient. If the log data sets are not the same size, it is more difficult to track your system's logs, and so space can be wasted.

The use of CSQJUFMT

Do not run a CSQJUFMT format when increasing the size of an active log.

If you run CSQJUFMT (in order to provide a performance advantage the first time the queue manager writes to the new active log) you receive messages:

```
IEC070I 203-204,XS95GTLX,REPRO02,OUTPUT,B857,SPMG02, 358
IEC070I MG.W.MG4E.LOGCOPY1.DS02,MG.W.MG4E.LOGCOPY1.DS02.DATA,
IDC3302I ACTION ERROR ON MG.W.MG4E.LOGCOPY1.DS02
IDC3351I ** VSAM I/O RETURN CODE IS 28 - RPLFDBWD = X'2908001C'
IDC31467I MAXIMUM ERROR LIMIT REACHED.

IDC0005I NUMBER OF RECORDS PROCESSED WAS 0
```

In addition, if you use the Access Method Services REPRO, ensure that you define a new empty log.

If you use REPRO to copy the old (renamed) data set into its respective new data set, the default is NOREPLACE.

This means that REPRO does not replace a record that is already on the designated data set. When formatting is done on the data set, the RBA value is reset. The net result is a data set that is not empty after formatting.

Changes for archive logs:

Use this topic to understand how to change the archive logs.

You can add to, delete from, and change the password of, entries in the BSDS for archive logs. Examples only are shown here; replace the data set names shown with the ones you want to use. For more details of the utility, see The change log inventory utility.

- Adding an archive log
- Deleting an archive log
- Changing the password of an archive log

Adding an archive log

When the recovery of an object depends on reading an existing archive log data set, the BSDS must contain information about that data set so that IBM MQ can find it. To register information about an existing archive log data set in the BSDS, use:

```
NEWLOG DSNAME=CSQARC1.ARCHLOG1.E00021.T2205197.A0000015,COPY1VOL=CSQV04,
UNIT=TAPE,STARTRBA=3A190000,ENDRBA=3A1F0FFF,CATALOG=NO
```

Deleting an archive log

To delete an entire archive log data set on one or more volumes, use:

```
DELETE DSNAME=CSQARC1.ARCHLOG1.E00021.T2205197.A0000015,COPY1VOL=CSQV04
```

Changing the password of an archive log

If you change the password of an existing archive log data set, you must also change the information in the BSDS.

1. List the BSDS, using the print log map utility.
2. Delete the entry for the archive log data set with the changed password, using the DELETE function of the CSQJU003 utility (see topic The change log inventory utility).
3. Name the data set as for a new archive log data set. Use the NEWLOG function of the CSQJU003 utility (see topic The change log inventory utility), and give the new password,

the starting and ending RBAs, and the volume serial numbers (which can be found in the print log map utility output, see The print log map utility).

To change the password for new archive log data sets, use:

```
ARCHIVE PASSWORD= password
```

To stop placing passwords on new archive log data sets, use:

```
ARCHIVE NOPASSWD
```

Note: Only use the ARCHIVE utility function if you do not have an external security manager.

Changing the high-level qualifier (HLQ) for the logs and BSDS:

Use this topic to understand the procedure required to change the high-level qualifier (HLQ).

Before you begin

You must end the queue manager normally before copying any of the logs or data sets to the new data sets. This is to ensure that the data is consistent and no recovery is needed during restart.

About this task

This task provides information about how to change the HLQ for the logs and BSDS. To do this, follow these steps:

Procedure

1. Run the log print utility CSQJU004 to record the log data set information. This information is needed later.
2. You can either:
   a. run DSS backup and restore with rename on the log and BSDS data sets to be renamed, or
   b. use AMS DEFINE and REPRO to create the HLQ data sets and copy the data from the old data sets.
3. Modify the MSTR and CHIN procedures to point to the new data sets.
4. Delete the old log information in the new copy of the BSDS using CSQJU003.
5. Define the new log data sets to the new BSDS using the NEWLOG function of CSQJU003. Keep all information about each log the same, apart from the HLQ.
6. The new BSDS should reflect the same information that was recorded for the old logs in the old BSDS. The HLQ should be the only thing that has changed.

What to do next

Compare the CSQJU004 output for the old and new BSDS to ensure that they look EXACTLY the same (except for the HLQs) before starting the queue manager.

Note: Care must be taken when performing these operations. Incorrect actions might lead to unrecoverable situations. Check the PRINT LOG MAP UTILITY output and make sure that all the information needed for recovery or restart has been included.

Recovering the BSDS:

If IBM MQ is operating in dual BSDS mode and one BSDS becomes damaged, forcing IBM MQ into single BSDS mode, IBM MQ continues to operate without a problem (until the next restart).

To return the environment to dual BSDS mode:

1. Use Access Method Services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the damaged BSDS. Example control statements can be found in job CSQ4BREC in thlqual.SCSQPROC.
2. Issue the IBM MQ command RECOVER BSDS to make a copy of the valid BSDS in the newly allocated data set and to reinstate dual BSDS mode.

If IBM MQ is operating in single BSDS mode and the BSDS is damaged, or if IBM MQ is operating in dual BSDS mode and both BSDSs are damaged, the queue manager stops and does not restart until the BSDS data sets are repaired. In this case:

1. Locate the BSDS associated with the most recent archive log data set. The data set name of the most recent archive log appears on the job log in the last occurrence of message CSQJ003I, which indicates that offload processing has been completed successfully. In preparation for the rest of this procedure, it is a good practice to keep a log of all successful archives noted by that message:
   - If archive logs are on DASD, the BSDS is allocated on any available DASD. The BSDS name is like the corresponding archive log data set name; change only the first letter of the last qualifier, from A to B, as in this example:

     Archive log name
     CSQ.ARCHLOG1. ***A*** 0000001

     BSDS copy name
     CSQ.ARCHLOG1. ***B*** 0000001
   - If archive logs are on tape, the BSDS is the first data set of the first archive log volume. The BSDS is not repeated on later volumes.
2. If the most recent archive log data set has no copy of the BSDS (for example, because an error occurred when offloading it), locate an earlier copy of the BSDS from earlier offload processing.
3. Rename *damaged* BSDSs using the Access Method Services ALTER command with the NEWNAME option. If you want to delete a damaged BSDS, use the Access Method Services DELETE command. For each damaged BSDS, use Access Method Services to define a new BSDS as a replacement data set. Job CSQ4BREC in thlqual.SCSQPROC contains Access Method Services control statements to define a new BSDS.
4. Use the Access Method Services REPRO command to copy the BSDS from the archive log to one of the replacement BSDSs you defined in step 3. Do not copy any data to the second replacement BSDS, you do that in step 5 on page 316.
   a. Print the contents of the replacement BSDS.

      Use the print log map utility (CSQJU004) to print the contents of the replacement BSDS. This enables you to review the contents of the replacement BSDS before continuing your recovery work.
   b. Update the archive log data set inventory in the replacement BSDS.

      Examine the output from the print log map utility and check that the replacement BSDS does not contain a record of the archive log from which the BSDS was copied. If the replacement BSDS is an old copy, its inventory might not contain all archive log data sets that were created more recently. The BSDS inventory of the archive log data sets must be updated to reflect the current subsystem inventory.

      Use the change log inventory utility (CSQJU003) NEWLOG statement to update the replacement BSDS, adding a record of the archive log from which the BSDS was copied. If the archive log data set is password-protected, use the PASSWORD option of the NEWLOG function. Also, if the archive log data set is cataloged, ensure that the CATALOG option of the NEWLOG function is properly set to CATALOG=YES. Use the NEWLOG statement to add any additional archive log data sets that were created later than the BSDS copy.

c. Update passwords in the replacement BSDS.

   The BSDS contains passwords for the archive log data sets and for the active log data sets. To ensure that the passwords in the replacement BSDS reflect the current passwords used by your installation, use the change log inventory ARCHIVE utility function with the PASSWORD option.

d. Update the active log data set inventory in the replacement BSDS.

   In unusual circumstances, your installation might have added, deleted, or renamed active log data sets since the BSDS was copied. In this case, the replacement BSDS does not reflect the actual number or names of the active log data sets your installation currently has in use.

   If you need to delete an active log data set from the replacement BSDS log inventory, use the change log inventory utility DELETE function.

   If you need to add an active log data set to the replacement BSDS log inventory, use the change log inventory utility NEWLOG function. Ensure that the RBA range is specified correctly on the NEWLOG function. If the active log data set is password-protected, use the PASSWORD option.

   If you need to rename an active log data set in the replacement BSDS log inventory, use the change log inventory utility DELETE function, followed by the NEWLOG function. Ensure that the RBA range is specified correctly on the NEWLOG function. If the active log data set is password-protected, use the PASSWORD option.

e. Update the active log RBA ranges in the replacement BSDS.

   Later, when the queue manager restarts, it compares the RBAs of the active log data sets listed in the BSDS with the RBAs found in the actual active log data sets. If the RBAs do not agree, the queue manager does not restart. The problem is magnified when an old copy of the BSDS is used. To solve this problem, use the change log inventory utility (CSQJU003) to adjust the RBAs found in the BSDS using the RBAs in the actual active log data sets. You do this by:

   - Using the print log records utility (CSQ1LOGP) to print a summary report of the active log data set. This shows the starting and ending RBAs.
   - Comparing the actual RBA ranges with the RBA ranges you have just printed, when the RBAs of all active log data sets are known.

     If the RBA ranges are equal for all active log data sets, you can proceed to the next recovery step without any additional work.

     If the RBA ranges are not equal, adjust the values in the BSDS to reflect the actual values. For each active log data set that needs to have the RBA range adjusted, use the change log inventory utility DELETE function to delete the active log data set from the inventory in the replacement BSDS. Then use the NEWLOG function to redefine the active log data set to the BSDS. If the active log data sets are password-protected, use the PASSWORD option of the NEWLOG function.

f. If only two active log data sets are specified for each copy of the active log, IBM MQ can have difficulty during queue manager restart. The problem can arise when one of the active log data sets is full and has not been offloaded, while the second active log data set is close to filling. In this case, add a new active log data set for each copy of the active log and define each new active log data set in the replacement BSDS log inventory.

   Use the Access Method Services DEFINE command to define a new active log data set for each copy of the active log and use the change log inventory utility NEWLOG function to define the new active log data sets in the replacement BSDS. You do not need to specify the RBA ranges on the NEWLOG statement. However, if the active log data sets are password-protected, use the PASSWORD option of the NEWLOG function. Example control statements to accomplish this task can be found in job CSQ4LREC in thlqual.SCSQPROC.

5. Copy the updated BSDS to the second new BSDS data set. The BSDSs are now identical.

   Use the print log map utility (CSQJU004) to print the contents of the second replacement BSDS at this point.

6. See Active log problems for information about what to do if you have lost your current active log data set.
7. Restart the queue manager using the newly constructed BSDS. IBM MQ determines the current RBA and what active logs need to be archived.

Managing page sets

Use this topic to understand how to manage the page sets associated with a queue manager.

This topic describes how to add, copy, and generally manage the page sets associated with a queue manager. It contains these sections:
- "How to change the high-level qualifier (HLQ) for the page sets"
- "How to add a page set to a queue manager"
- "What to do when one of your page sets becomes full" on page 318
- "How to balance loads on page sets" on page 318
- How to increase the size of a page set
- "How to reduce a page set" on page 322
- "How to reintroduce a page set" on page 322
- "How to back up and recover page sets" on page 323
- "How to delete page sets" on page 327
- "How to back up and restore queues using CSQUTIL" on page 327

See Page sets for a description of page sets, storage classes, buffers, and buffer pools, and some of the performance considerations that apply.

How to change the high-level qualifier (HLQ) for the page sets

This task gives information on how to change the HLQ for the page sets. To perform this task, do the following:
1. Define the new HLQ page sets.
2. If the size allocation is the same as the old page sets, copy the existing page set using REPRO to the empty new HLQ page sets. If you are increasing the size of the page sets, use the FORMAT function of CSQUTIL to format the destination page set. For more information, see Formatting page sets (FORMAT).
3. Use the COPYPAGE function of CSQUTIL to copy all the messages from the source page set to the destination page set. For more information, see Expanding a page set (COPYPAGE).
4. Change the CSQP00xx DD statement in the queue manager procedure to point to the new HLQ page sets.

Restart the queue manager and verify the changes to the page sets.

How to add a page set to a queue manager

This description assumes that you have a queue manager that is already running. You might need to add a page set if, for example, your queue manager has to cope with new applications using new queues.

To add a new page set, use the following procedure:
1. Define and format the new page set. You can use the sample JCL in thlqual.SCSQPROC(CSQ4PAGE) as a basis. For more information, see Formatting page sets (FORMAT).

   Take care not to format any page sets that are in use, unless this is what you intend. If so, use the FORCE option of the FORMAT utility function.
2. Use the DEFINE PSID command with the DSN option to associate the page set with a buffer pool.

3. Add the appropriate storage class definitions for your page set by issuing DEFINE STGCLASS commands.
4. Optionally, to document how your queue manager is configured:
   a. Add the new page set to the started task procedure for your queue manager.
   b. Add a definition for the new page set to your CSQINP1 initialization data set.
   c. Add a definition for the new storage class to your CSQ4INYR initialization data set member.

For details of the DEFINE PSID and DEFINE STGCLASS commands, see DEFINE PSID and DEFINE STGCLASS.

What to do when one of your page sets becomes full

You can find out about the utilization of page sets by using the IBM MQ command DISPLAY USAGE. For example, the command:

```
DISPLAY USAGE PSID(03)
```

displays the current state of the page set 03. This tells you how many free pages this page set has.

If you have defined secondary extents for your page sets, they are dynamically expanded each time they fill up. Eventually, all secondary extents are used, or no further disk space is available. If this happens, an application receives the return code MQRC_STORAGE_MEDIUM_FULL.

If an application receives a return code of MQRC_STORAGE_MEDIUM_FULL from an MQI call, this is a clear indication that there is not enough space left on the page set. If the problem persists or is likely to recur, you must do something to solve it.

You can approach this problem in a number of ways:
- Balance the load between page sets by moving queues from one page set to another.
- Expand the page set. See “How to increase the size of a page set” on page 320 for instructions.
- Redefine the page set so that it can expand beyond 4 GB to a maximum size of 64 GB. See Defining a page set to be larger than 4 GB for instructions.

How to balance loads on page sets

Load balancing on page sets means moving the messages associated with one or more queues from one page set to another, less used, page set. Use this technique if it is not practical to expand the page set.

To identify which queues are using a page set, use the appropriate IBM MQ commands. For example, to find out which queues are mapped to page set 02, first, find out which storage classes map to page set 02, by using the command:

```
DISPLAY STGCLASS(*) PSID(02)
```

Then use the following command to find out which queues use which storage class:

```
DISPLAY QUEUE(*) TYPE(QLOCAL) STGCLASS
```

Moving a non-shared queue

To move queues and their messages from one page set to another, use the MQSC MOVE QLOCAL command (described in MOVE QLOCAL). When you have identified the queue or queues that you want to move to a new page set, follow this procedure for each of these queues:

1. Ensure that the queue you want to move is not in use by any applications (that is, IPPROCS and OPPROCS values from the DISPLAY QSTATUS command are zero) and that it has no uncommitted messages (the UNCOM value from the DISPLAY QSTATUS command is NO).

   Note: The only way to ensure that this state continues is to change your security settings temporarily. If you cannot do this, later stages in this procedure might fail if applications start to use the queue despite precautionary steps such as setting PUT(DISABLED). However, messages can never be lost by this procedure.
2. Prevent applications from putting messages on the queue being moved by altering the queue definition to disable **MQPUT** s. Change the queue definition to PUT(DISABLED).
3. Define a temporary queue with the same attributes as the queue that is being moved, using the command:

```
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
```

   Note: If this temporary queue already exists from a previous run, delete it before doing the define.
4. Move the messages to the temporary queue using the following command:

```
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
```

5. Delete the queue you are moving, using the command:

```
DELETE QLOCAL(QUEUE_TO_MOVE)
```

6. Define a new storage class that maps to the required page set, for example:

```
DEFINE STGCLASS(NEW) PSID(nn)
```

   Add the new storage class definition to the CSQINP2 data sets ready for the next queue manager restart.
7. Redefine the queue that you are moving, by changing the storage class attribute:

```
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) STGCLASS(NEW)
```

When the queue is redefined, it is based on the temporary queue created in step 3 on page 319.

8. Move the messages back to the new queue, using the command:

```
MOVE QLOCAL(TEMP) TOQLOCAL(QUEUE_TO_MOVE)
```

9. The queue created in step 3 on page 319 is no longer required. Use the following command to delete it:

```
DELETE QL(TEMP_QUEUE)
```

10. If the queue being moved was defined in the CSQINP2 data sets, change the STGCLASS attribute of the appropriate DEFINE QLOCAL command in the CSQINP2 data sets. Add the REPLACE keyword so that the existing queue definition is replaced.

Figure 44 shows an extract from a load balancing job.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE STGCLASS(NEW) PSID(2)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) STGCLASS(NEW)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 44. Extract from a load balancing job for a page set

How to increase the size of a page set

You can initially allocate a page set larger than 4 GB, See Defining a page set to be larger than 4 GB

A page set can be defined to be automatically expanded as it becomes full by specifying EXPAND(SYSTEM) or EXPAND(USER). If your page set was defined with EXPAND(NONE), you can expand it in either of two ways:

- Alter its definition to allow automatic expansion. See Altering a page set to allow automatic expansion
- Create a new, larger page set and copy the messages from the old page set to the new one. See Moving messages to a new, larger page set

Defining a page set to be larger than 4 GB

IBM MQ can use a page set up to 64 GB in size, provided the data set is defined with 'extended addressability' to VSAM. Extended addressability is an attribute which is conferred by an SMS data class. In the example shown in the following sample JCL, the management class 'EXTENDED' is defined to SMS with 'Extended addressability'. If your existing page set is not currently defined as having extended addressability, use the following method to migrate to an extended addressability format data set.

1. Stop the queue manager.
2. Use Access Method Services to rename the existing page set.
3. Define a destination page set, the same size as the existing page set, but with DATACLASS(EXTENDED).
4. Use the COPYPAGE function of CSQUTIL to copy all the messages from the source page set to the destination page set. See Expanding a page set (COPYPAGE) for more details.
5. Restart the queue manager.
6. Alter the page set to use system expansion, to allow it to continue growing beyond its current allocation.

The following JCL shows example Access Method Services commands:

```
//S1    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN   DD  *
ALTER 'VICY.CSQ1.PAGE01' -
NEWNAME('VICY.CSQ1.PAGE01.OLD')
ALTER 'VICY.CSQ1.PAGE01.DATA' -
NEWNAME('VICY.CSQ1.PAGE01.DATA.OLD')
DEFINE CLUSTER (NAME('VICY.CSQ1.PAGE01') -
MODEL('VICY.CSQ1.PAGE01.OLD') -
DATACLASS(EXTENDED))
/*
```

Altering a page set to allow automatic expansion

Use the ALTER PSID command with the EXPAND(USER) or EXPAND(SYSTEM) options. See ALTER PSID and Expanding a page set (COPYPAGE) for general information on expanding page sets.

Moving messages to a new, larger page set

This technique involves stopping and restarting the queue manager. This deletes any nonpersistent messages that are not on shared queues at restart time. If you have nonpersistent messages that you do not want to be deleted, use load balancing instead. For more details, see “How to balance loads on page sets” on page 318. In this description, the page set that you want to expand is referred to as the *source* page set; the new, larger page set is referred to as the *destination* page set.

Follow these steps:

1. Stop the queue manager.
2. Define the destination page set, ensuring that it is larger than the source page set, with a larger secondary extent value.
3. Use the FORMAT function of CSQUTIL to format the destination page set. See Formatting page sets (FORMAT) for more details.
4. Use the COPYPAGE function of CSQUTIL to copy all the messages from the source page set to the destination page set. See Expanding a page set (COPYPAGE) for more details.
5. Restart the queue manager using the destination page set by doing one of the following:
   - Change the queue manager started task procedure to reference the destination page set.

- Use Access Method Services to delete the source page set and then rename the destination page set, giving it the same name as that of the source page set.

Attention:

Before you delete any IBM MQ page set, be sure that you have made the required backup copies.

How to reduce a page set

Prevent all users, other than the IBM MQ administrator, from using the queue manager. For example; by changing the access security settings.

If you have a large page set that is mostly empty (as shown by the DISPLAY USAGE command), you might want to reduce its size. The procedure to do this involves using the COPY, FORMAT, and LOAD functions of CSQUTIL (see IBM MQ utility program). This procedure does not work for page set zero (0), as it is not practical to reduce the size of this page set; the only way to do so is by reinitializing your queue manager (see “Reinitializing a queue manager” on page 340). The prerequisite of this procedure is to try and remove all users from the system so that all UOWs are complete and the page sets are consistent.

1. Use the STOP QMGR command with the `QUIESCE` or `FORCE` attribute to stop the queue manager.
2. Run the SCOPY function of CSQUTIL with the `PSID` option, to copy all message data from the large page set and save them in a sequential data set.
3. Define a new smaller page set data set to replace the large page set.
4. Run the FORMAT TYPE(NEW) function of CSQUTIL against the page set that you created in step 3.
5. Restart the queue manager using the page set created in step 3.
6. Run the LOAD function of CSQUTIL to load back all the messages saved during step 2.
7. Allow all users access to the queue manager.
8. Delete the old large page set.

How to reintroduce a page set

In certain scenarios it is useful to be able to bring an old page set online again to the queue manager. Unless specific action is taken, when the old page set is brought online the queue manager will recognize that the page set recovery RBA stored in the page set itself and in the checkpoint records is old, and will therefore automatically start media recovery of the page set to bring it up to date.

Such media recovery can only be performed at queue manager restart, and is likely to take a considerable length of time, especially if archive logs held on tape must be read. However, normally in this circumstance, the page set has been offline for the intervening period and so the log contains no information pertinent to the page set recovery.

The following three choices are available:

Allow full media recovery to be performed.

1. Stop the queue manager.
2. Ensure definitions are available for the page set in both the started task procedure for the queue manager and in the CSQINP1 initialization data set.
3. Restart the queue manager.

Allow any messages on the page set to be destroyed.

This choice is useful where a page set has been offline for a long time (some months, for example) and it has now been decided to reuse it for a different purpose.

1. Format the page set using the FORMAT function of CSQUTIL with the TYPE(NEW) option.

2. Add definitions for the page set to both the started task procedure for the queue manager and the CSQINP1 initialization data set.
3. Restart the queue manager.

Using the TYPE(NEW) option for formatting clears the current contents of the page set and tells the queue manager to ignore any historical information in the checkpoint about the page set.

Bring the page set online avoiding the media recovery process.
Use this technique only if you are sure that the page set has been offline since a clean shutdown of the queue manager. This choice is most appropriate where the page set has been offline for a short period, typically due to operational issues such as a backup running while the queue manager is being started.

1. Format the page set using the FORMAT function of CSQUTIL with the TYPE(REPLACE) option.
2. Either add the page set back into the queue manager dynamically using the DEFINE PSID command with the DSN option or allow it to be added at a queue manager restart.

Using the TYPE(REPLACE) option for formatting checks that the page set was cleanly closed by the queue manager, and marks it so that media recovery will not be performed. No other changes are made to the contents of the page set.

How to back up and recover page sets

There are different mechanisms available for back up and recovery. Use this topic to understand these mechanisms.

This section describes the following topics:

- "Creating a point of recovery for non-shared resources"
- "Backing up page sets" on page 324
- "Recovering page sets" on page 325
- How to delete page sets

For information about how to create a point of recovery for shared resources, see "Recovering shared queues" on page 331.

Creating a point of recovery for non-shared resources

IBM MQ can recover objects and non-shared persistent messages to their current state if both:

1. Copies of page sets from an earlier point exist.
2. All the IBM MQ logs are available to perform recovery from that point.

These represent a point of recovery for non-shared resources.

Both objects and messages are held on page sets. Multiple objects and messages from different queues can exist on the same page set. For recovery purposes, objects and messages cannot be backed up in isolation, so a page set must be backed up as a whole to ensure the correct recovery of the data.

The IBM MQ recovery log contains a record of all persistent messages and changes made to objects. If IBM MQ fails (for example, due to an I/O error on a page set), you can recover the page set by restoring the backup copy and restarting the queue manager. IBM MQ applies the log changes to the page set from the point of the backup copy.

There are two ways of creating a point of recovery:

Full backup
Stop the queue manager, which forces all updates on to the page sets.

This allows you to restart from the point of recovery, using only the backed up page set data sets and the logs from that point on.

Fuzzy backup

Take *fuzzy* backup copies of the page sets without stopping the queue manager.

If you use this method, and your associated logs later become damaged or lost, you cannot to use the fuzzy page set backup copies to recover. This is because the fuzzy page set backup copies contain an inconsistent view of the state of the queue manager and are dependent on the logs being available. If the logs are not available, you need to return to the last set of backup page set copies taken while the subsystem was inactive (Method 1) and accept the loss of data from that time.

Method 1: Full backup

This method involves shutting the queue manager down. This forces all updates on to the page sets so that the page sets are in a consistent state.

1. Stop all the IBM MQ applications that are using the queue manager (allowing them to complete first). This can be done by changing the access security or queue settings, for example.
2. When all activity has completed, display and resolve any in-doubt units of recovery. (Use the commands DISPLAY CONN and RESOLVE INDOUBT, as described in DISPLAY CONN and RESOLVE INDOUBT.)

   This brings the page sets to a consistent state; if you do not do this, your page sets might not be consistent, and you are effectively doing a fuzzy backup.
3. Issue the ARCHIVE LOG command to ensure that the latest log data is written out to the log data sets.
4. Issue the STOP QMGR MODE(QUIESCE) command. Record the lowest RBA value in the CSQI024I or CSQI025I messages (see CSQI024I and CSQI025I for more information). You should keep the log data sets starting from the one indicated by the RBA value up to the current log data set.
5. Take backup copies of all the queue manager page sets (see “Backing up page sets”).

Method 2: Fuzzy backup

This method does not involve shutting the queue manager down. Therefore, updates might be in virtual storage buffers during the backup process. This means that the page sets are not in a consistent state, and can only be used for recovery with the logs.

1. Issue the DISPLAY USAGE TYPE(ALL) command, and record the RBA value in the CSQI024I or CSQI025I messages (see CSQI024I and CSQI025I for more information).
2. Take backup copies of the page sets (see “Backing up page sets”).
3. Issue the ARCHIVE LOG command, to ensure that the latest log data is written out to the log data sets. To restart from the point of recovery, you must keep the log data sets starting from the log data set indicated by the RBA value up to the current log data set.

Backing up page sets

To recover a page set, IBM MQ needs to know how far back in the log to go. IBM MQ maintains a log RBA number in page zero of each page set, called the *recovery log sequence number* (LSN). This number is the starting RBA in the log from which IBM MQ can recover the page set. When you back up a page set, this number is also copied.

If the copy is later used to recover the page set, IBM MQ must have access to all the log records from this RBA value to the current RBA. That means you must keep enough of the log records to enable IBM MQ to recover from the oldest backup copy of a page set you intend to keep.

Use ADRDSSU COPY function to copy the page sets.

For more information, see the COPY DATASET Command Syntax for Logical Data Set documentation .

For example:

```
//STEP2 EXEC PGM=ADRDSSU,REGION=6M
//SYSPRINT DD SYSOUT=H
//SYSIN DD *
 COPY -
 DATASET(INCLUDE(SCENDATA.MQPA.PAGESET.*)) -
 RENAMEU(SCENDATA.MQPA.PAGESET.**,SCENDATA.MQPA.BACKUP1.**) -
 SPHERE -
 REPUNC -
 FASTREPLICATION(PREF )-
 CANCELERROR -
 TOL(ENQF)
/*
//
```

If you copy the page set while the queue manager is running you must use a copy utility that copies page zero of the page set first. If you do not do this you could corrupt the data in your page set.

If the process of dynamically expanding a page set is interrupted, for example by power to the system being lost, you can still use ADRDSSU to take a backup of a page set.

If you perform an Access Method Services IDCAMS `LISTCAT ENT('page set data set name') ALLOC`, you will see that the `HI-ALLOC-RBA` is higher than the `HI-USED-RBA`.

The next time this page set fills up it is extended again, if possible, and the pages between the high used RBA and the highest allocated RBA are used, along with another new extent.

Backing up your object definitions

You should also back up copies of your object definitions. To do this, use the MAKEDEF feature of the CSQUTIL COMMAND function (described in Issuing commands to IBM MQ (COMMAND)).

Back up your object definitions whenever you take a backup copy of your queue manager, and keep the most current version.

Recovering page sets

If the queue manager has terminated due to a failure, the queue manager can normally be restarted with all recovery being performed during restart. However, such recovery is not possible if any of your page sets or log data sets are not available. The extent to which you can now recover depends on the availability of backup copies of page sets and log data sets.

To restart from a point of recovery you must have:

- A backup copy of the page set that is to be recovered.
- If you used the “fuzzy” backup process described in “Method 2: Fuzzy backup” on page 324, the log data set that included the recorded RBA value, the log data set that was made by the ARCHIVE LOG command, and all the log data sets between these.
- If you used full backup, but you do not have the log data sets following that made by the ARCHIVE LOG command, you do **not** need to run the FORMAT TYPE(REPLACE) function of the CSQUTIL utility against all your page sets.

To recover a page set to its current state, you must also have all the log data sets and records since the ARCHIVE LOG command.

There are two methods for recovering a page set. To use either method, the queue manager must be stopped.

Simple recovery

This is the simpler method, and is appropriate for most recovery situations.

1. Delete the page set you want to restore from backup.
2. Use the ADRDSSU COPY function to recover your page set from the backup copy..

   Alternatively, you can rename your backup copy to the original name, or change the CSQP00xx DD statement in your queue manager procedure to point to your backup page set. However, if you then lose or corrupt the page set, you will no longer have a backup copy to restore from.
3. Restart the queue manager.
4. When the queue manager has restarted successfully, you can restart your applications
5. Reinstate your normal backup procedures for the restored page.

Advanced recovery

This method provides performance advantages if you have a large page set to recover, or if there has been much activity on the page set since the last backup copy was taken. However, it requires more manual intervention than the simple method, which might increase the risk of error and the time taken to perform the recovery.

1. Delete and redefine the page set you want to restore from backup.
2. Use ADRDSSU to copy the backup copy of the page set into the new page set. Define your new page set with a secondary extent value so that it can be expanded dynamically.

   Alternatively, you can rename your backup copy to the original name, or change the CSQP00xx DD statement in your queue manager procedure to point to your backup page set. However, if you then lose or corrupt the page set, you will no longer have a backup copy to restore from.
3. Change the CSQINP1 definitions for your queue manager to make the buffer pool associated with the page set being recovered as large as possible. By making the buffer pool large, you might be able to keep all the changed pages resident in the buffer pool and reduce the amount of I/O to the page set.
4. Restart the queue manager.
5. When the queue manager has restarted successfully, stop it (using quiesce) and then restart it using the normal buffer pool definition for that page set. After this second restart completes successfully, you can restart your applications
6. Reinstate your normal backup procedures for the restored page.

What happens when the queue manager is restarted

When the queue manager is restarted, it applies all changes made to the page set that are registered in the log, beginning at the restart point for the page set. IBM MQ can recover multiple page sets in this way. The page set is dynamically expanded, if required, during media recovery.

During restart, IBM MQ determines the log RBA to start from by taking the lowest value from the following:

- Recovery LSN from the checkpoint log record for each page set.
- Recovery LSN from page zero in each page set.
- The RBA of the oldest incomplete unit of recovery in the system at the time the backup was taken.

All object definitions are stored on page set zero. Messages can be stored on any available page set.

Note: The queue manager cannot restart if page set zero is not available.

How to delete page sets

You delete a page set by using the DELETE PSID command; see DELETE PSID for details of this command.

You cannot delete a page set that is still referenced by any storage class. Use DISPLAY STGCLASS to find out which storage classes reference a page set.

The data set is deallocated from IBM MQ but is not deleted. It remains available for future use, or can be deleted using z/OS facilities.

Remove the page set from the started task procedure for your queue manager.

Remove the definition of the page set from your CSQINP1 initialization data set.

How to back up and restore queues using CSQUTIL

Use this topic as a reference for further information about back up and restore using CSQUTIL.

You can use the CSQUTIL utility functions for backing up and restoring queues. To back up a queue, use the COPY or SCOPY function to copy the messages from a queue onto a data set. To restore the queue, use the complementary function LOAD or SLOAD. For more information, see IBM MQ utility program.

Managing buffer pools

Use this topic if you want to change or delete your buffer pools.

This topic describes how to alter and delete buffer pools. It contains these sections:
- “How to change the number of buffers in a buffer pool”
- “How to delete a buffer pool” on page 328

Buffer pools are defined during queue manager initialization, using DEFINE BUFFPOOL commands issued from the initialization input data set CSQINP1. Their attributes can be altered in response to business requirements while the queue manager is running, using the processes detailed in this topic. The queue manager records the current buffer pool attributes in checkpoint log records. These are automatically restored on subsequent queue manager restart, unless the buffer pool definition in CSQINP1 includes the REPLACE attribute.

Use the DISPLAY USAGE command to display the current buffer attributes.

You can also define buffer pools dynamically using the DEFINE PSID command with the DSN option.

If you change buffer pools dynamically, you should also update their definitions in the initialization data set CSQINP1.

See Planning on z/OS for a description of page sets, storage classes, buffers, and buffer pools, and some of the performance considerations that apply.

Note: Buffer pools use significant storage. When you increase the size of a buffer pool or define a new buffer pool ensure that sufficient storage is available. For more information, see Address space storage.

How to change the number of buffers in a buffer pool

If a buffer pool is too small, the condition can result in message CSQP020E on the console, you can allocate more buffers to it using the ALTER BUFFPOOL command as follows:

1. Determine how much space is available for new buffers by looking at the CSQY220I messages in the log. The available space is reported in MB. As a buffer has a size of 4 KB, each MB of available space allows you to allocate 256 buffers. Do not allocate all the free space to buffers, as some is required for other tasks.

   If the buffer pool uses fixed 4 KB pages, that is, its PAGECLAS attribute is FIXED4KB, ensure that there is sufficient real storage available on the LPAR.
2. If the reported free space is inadequate, release some buffers from another buffer pool using the command

   ```
   ALTER BUFFPOOL(buf-pool-id) BUFFERS(integer)
   ```

   where *buf-pool-id* is the buffer pool from which you want to reclaim space and *integer* is the new number of buffers to be allocated to this buffer pool, which must be smaller than the original number of buffers allocated to it.
3. Add buffers to the buffer pool you want to expand using the command

   ```
   ALTER BUFFPOOL(buf-pool-id) BUFFERS(integer)
   ```

   where *buf-pool-id* is the buffer pool to be expanded and *integer* is the new number of buffers to be allocated to this buffer pool, which must be larger than the original number of buffers allocated to it.

How to delete a buffer pool

When a buffer pool is no longer used by any page sets, delete it to release the virtual storage allocated to it.

You delete a buffer pool using the DELETE BUFFPOOL command. The command fails if any page sets are using this buffer pool.

See "How to delete page sets" on page 327 for information about how to delete page sets.

Managing queue-sharing groups and shared queues

IBM MQ can use different types of shared resources, for example queue-sharing groups, shared queues, and the coupling facility. Use this topic to review the procedures needed to manage these shared resources.

This section contains information about the following topics:
- "Managing queue-sharing groups" on page 329
- "Managing shared queues" on page 331
- "Managing group objects" on page 336
- "Managing the coupling facility" on page 336

Managing queue-sharing groups:

You can add or remove a queue manager to a queue-sharing group, and manage the associated Db2 tables.

This topic has sections about the following tasks:
- “Adding a queue-sharing group to the Db2 tables”
- “Adding a queue manager to a queue-sharing group”
- “Removing a queue manager from a queue-sharing group”
- “Removing a queue-sharing group from the Db2 tables” on page 330
- “Validating the consistency of Db2 definitions” on page 330

Adding a queue-sharing group to the Db2 tables

To add a queue-sharing group to the Db2 tables, use the ADD QSG function of the queue-sharing group utility (CSQ5PQSG). This program is described in The queue-sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45AQS).

Adding a queue manager to a queue-sharing group

A queue manager can be added to a queue-sharing group and this topic describes some of the limitations.

To add a queue manager to a queue-sharing group, use the ADD QMGR function of the queue-sharing group utility (CSQ5PQSG). This program is described in The queue-sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45AQM).

The queue-sharing group must exist before you can add queue managers to it.

Queue-sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

A queue manager can only be a member of one queue-sharing group.

Note: To add a queue manager to an existing queue-sharing group containing queue managers running earlier versions of IBM MQ, you must first apply the coexistence PTF for the highest version of IBM MQ in the group to every earlier version queue manager in the group.

Removing a queue manager from a queue-sharing group

You can only remove a queue manager from a queue-sharing group if the queue manager's logs are not needed by another process. The logs are needed if they contain:
- the latest backup of one of the coupling facility (CF) application structures used by the queue-sharing group
- data needed by a future restore process, that is, the queue manager has used a recoverable structure since the time described by the last backup exclusion interval value.

If either or both of these points apply, the queue manager cannot be removed. To determine which queue managers' logs are needed for a future restore process, use the MQSC DISPLAY CFSTATUS command with the TYPE(BACKUP) option (for details of this command, see DISPLAY CFSTATUS).

If the queue manager logs are not needed, take the following steps to remove the queue manager from the queue-sharing group:
1. Resolve any indoubt units of work involving this queue manager.

2. Shut the queue manager down cleanly using STOP QMGR MODE(QUIESCE).
3. Wait for an interval at least equivalent to the value of the EXCLINT parameter you will specify in the BACKUP CFSTRUCT command in the next step.
4. On another queue manager, run a CF structure backup for each recoverable CF structure by using the MQSC BACKUP CFSTRUCT command and specifying an EXCLINT value as required in the previous step.
5. Use the REMOVE QMGR function of the CSQ5PQSG utility to remove the queue manager from the queue-sharing group. This program is described in The queue-sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45RQM).
6. Before restarting the queue manager, reset the QSGDATA system parameter to its default value. See Using CSQ6SYSP for information about how to tailor your system parameters.

Note, that when removing the last queue manager in a queue sharing group, you must use the FORCE option, rather than REMOVE. This removes the queue manager from the queue sharing group, while not performing the consistency checks of the queue manager logs being required for recovery. You should only perform this operation if you are deleting the queue sharing group.

Removing a queue-sharing group from the Db2 tables

To remove a queue-sharing group from the Db2 tables, use the REMOVE QSG function of the queue-sharing group utility (CSQ5PQSG). This program is described in The queue-sharing group utility. A sample is provided in thlqual.SCSQPROC(CSQ45RQS).

You can only remove a queue-sharing group from the common Db2 data-sharing group tables after you have removed all the queue managers from the queue-sharing group (as described in "Removing a queue manager from a queue-sharing group" on page 329).

When the queue-sharing group record is deleted from the queue-sharing group administration table, all objects and administrative information relating to that queue-sharing group are deleted from other IBM MQ Db2 tables. This includes shared queue and group object information.

Validating the consistency of Db2 definitions

Problems for shared queues within a queue-sharing group can occur if the Db2 object definitions have, for any reason, become inconsistent.

To validate the consistency of the Db2 object definitions for queue managers, CF structures, and shared queues, use the VERIFY QSG function of the queue-sharing group utility (CSQ5PQSG). This program is described in The queue-sharing group utility.

Managing shared queues:

Use this topic to understand how to recover, move, and migrate shared queues.

This section describes the following tasks:
- “Recovering shared queues”
- “Moving shared queues”
- “Migrating non-shared queues to shared queues” on page 334
- Suspending a Db2 connection

Recovering shared queues

IBM MQ can recover persistent messages on shared queues if all:
- Backups of the CF structures containing the messages have been performed.
- All the logs for all queue managers in the queue-sharing group are available, to perform recovery from the point the backups are taken.
- Db2 is available and the structure backup table is more recent that the most recent CF structure backup.

The messages on a shared queue are stored in a coupling facility (CF) structure. Persistent messages can be put onto shared queues, and like persistent messages on non-shared queues, they are copied to the queue manager log. The MQSC BACKUP CFSTRUCT and RECOVER CFSTRUCT commands are provided to allow the recovery of a CF structure in the unlikely event of a coupling facility failure. In such circumstances, any nonpersistent messages stored in the affected structure are lost, but persistent messages can be recovered. Any further application activity using the structure is prevented until the structure has been recovered.

To enable recovery, you must back up your coupling facility list structures frequently using the MQSC BACKUP CFSTRUCT command. The messages in the CF structure are written onto the active log data set of the queue manager making the backup. It writes a record of the backup to Db2: the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager log, and the backup time. Back up CF list structures even if you are not actively using shared queues, for example, if you have set up a queue-sharing group intending to use it in the future.

You can recover a CF structure by issuing an MQSC RECOVER CFSTRUCT command to the queue manager that can perform the recovery; you can use any queue manager in the queue-sharing group. You can specify a single CF structure to be recovered, or you can recover several CF structures simultaneously.

As noted previously, it is important that you back up your CF list structures frequently, otherwise recovering a CF structure can take a long time. Moreover, the recovery process cannot be canceled.

The definition of a shared queue is kept in a Db2 database and can therefore be recovered if necessary using standard Db2 database procedures. See Shared queues and queue-sharing groups for more information.

Moving shared queues

This section describes how to perform load balancing by moving a shared queue from one coupling facility structure to another. It also describes how to move a non-shared queue to a shared queue, and how to move a shared queue to a non-shared queue.

When you move a queue, you need to define a temporary queue as part of the procedure. This is because every queue must have a unique name, so you cannot have two queues of the same name, even if the queues have different queue dispositions. IBM MQ tolerates having two queues with the same name (as in step 2), but you cannot use the queues.

- Moving a queue from one coupling facility structure to another
- Moving a non-shared queue to a shared queue
- Moving a shared queue to a non-shared queue

Moving a queue from one coupling facility structure to another

To move queues and their messages from one CF structure to another, use the MQSC MOVE QLOCAL command. When you have identified the queue or queues that you want to move to a new CF structure, use the following procedure to move each queue:

1. Ensure that the queue you want to move is not in use by any applications, that is, the queue attributes IPPROCS and OPPROCS are zero on all queue managers in the queue-sharing group.
2. Prevent applications from putting messages on the queue being moved by altering the queue definition to disable **MQPUT** s. Change the queue definition to PUT(DISABLED).
3. Define a temporary queue with the same attributes as the queue that is being moved using the following command:

```
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED) QSGDISP(QMGR)
```

   Note: If this temporary queue exists from a previous run, delete it before doing the define.
4. Move the messages to the temporary queue using the following command:

```
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
```

5. Delete the queue you are moving, using the command:

```
DELETE QLOCAL(QUEUE_TO_MOVE)
```

6. Redefine the queue that is being moved, changing the CFSTRUCT attribute, using the following command:

```
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
```

   When the queue is redefined, it is based on the temporary queue created in step 3.
7. Move the messages back to the new queue using the command:

```
MOVE QLOCAL(TEMP) TOQLOCAL(QUEUE_TO_MOVE)
```

8. The queue created in step 3 on page 332 is no longer required. Use the following command to delete it:

```
DELETE QL(TEMP_QUEUE)
```

9. If the queue being moved was defined in the CSQINP2 data sets, change the CFSTRUCT attribute of the appropriate DEFINE QLOCAL command in the CSQINP2 data sets. Add the REPLACE keyword so that the existing queue definition is replaced.

Figure 45 shows a sample job for moving a queue from one CF structure to another.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//      DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED) QSGDISP(QMGR)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 45. Sample job for moving a queue from one CF structure to another

Moving a non-shared queue to a shared queue

The procedure for moving a non-shared queue to a shared queue is like the procedure for moving a queue from one CF structure to another (see “Moving a queue from one coupling facility structure to another” on page 332). Figure 46 on page 334 gives a sample job to do this.

Note: Remember that messages on shared queues are subject to certain restrictions on the maximum message size, message persistence, and queue index type, so you might not be able to move some non-shared queues to a shared queue.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 46. Sample job for moving a non-shared queue to a shared queue

Moving a shared queue to a non-shared queue

The procedure for moving a shared queue to a non-shared queue is like the procedure for moving a queue from one CF structure to another (see "Moving a queue from one coupling facility structure to another" on page 332).

Figure 47 gives a sample job to do this.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
ALTER QL(QUEUE_TO_MOVE) PUT(DISABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED) QSGDISP(QMGR)
MOVE QLOCAL(QUEUE_TO_MOVE) TOQLOCAL(TEMP_QUEUE)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) STGCLASS(NEW) QSGDISP(QMGR)
MOVE QLOCAL(TEMP_QUEUE) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 47. Sample job for moving a shared queue to a non-shared queue

Migrating non-shared queues to shared queues

There are two stages to migrating non-shared queues to shared queues:
- Migrating the first (or only) queue manager in the queue-sharing group
- Migrating any other queue managers in the queue-sharing group

Migrating the first (or only) queue manager in the queue-sharing group

Figure 46 shows an example job for moving a non-shared queue to a shared queue. Do this for each queue that needs migrating.

Note:

1. Messages on shared queues are subject to certain restrictions on the maximum message size, message persistence, and queue index type, so you might not be able to move some non-shared queues to a shared queue.
2. You must use the correct index type for shared queues. If you migrate a transmission queue to be a shared queue, the index type must be MSGID.

If the queue is empty, or you do not need to keep the messages that are on it, migrating the queue is simpler. Figure 48 shows an example job to use in these circumstances.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD  DSN=thlqual.SCSQANLE,DISP=SHR
//     DD  DSN=thlqual.SCSQAUTH,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN  DD  *
COMMAND DDNAME(MOVEQ)
/*
//MOVEQ DD  *
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(QUEUE_TO_MOVE) PUT(ENABLED) GET(ENABLED)
DELETE QL(QUEUE_TO_MOVE)
DEFINE QL(QUEUE_TO_MOVE) LIKE(TEMP_QUEUE) CFSTRUCT(NEW) QSGDISP(SHARED)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 48. Sample job for moving a non-shared queue without messages to a shared queue

Migrating any other queue managers in the queue-sharing group

1. For each queue that does not have the same name as an existing shared queue, move the queue as described in Figure 46 on page 334 or Figure 48.
2. For queues that have the same name as an existing shared queue, move the messages to the shared queue using the commands shown in Figure 49.

```
MOVE QLOCAL(QUEUE_TO_MOVE) QSGDISP(QMGR) TOQLOCAL(QUEUE_TO_MOVE)
DELETE QLOCAL(QUEUE_TO_MOVE) QSGDISP(QMGR)
```

Figure 49. Moving messages from a non-shared queue to an existing shared queue

Suspending a connection to Db2

If you want to apply maintenance or service to the Db2 tables or package related to shared queues without stopping your queue manager, you must temporarily disconnect queue managers in the data sharing group (DSG) from Db2.

To do this:

1. Use the MQSC command SUSPEND QMGR FACILITY(Db2).
2. Do the binds.
3. Reconnect to Db2 using the MQSC command RESUME QMGR FACILITY(Db2)

Note that there are restrictions on the use of these commands.

Attention: While the Db2 connection is suspended, the following operations will not be available. Therefore, you need to do this work during a time when your enterprise is at its least busy.

- Access to Shared queue objects for administration (define, delete,alter)
- Starting shared channels
- Storing messages in Db2
- Backup or recover CFSTRUCT

Managing group objects:

Use this topic to understand how to work with group objects.

IBM MQ automatically copies the definition of a group object to page set zero of each queue manager that uses it. You can alter the copy of the definition temporarily, and IBM MQ allows you to refresh the page set copies from the repository copy. IBM MQ always tries to refresh the page set copies from the repository copy on start-up (for channel objects, this is done when the channel initiator restarts). This ensures that the page set copies reflect the version on the repository, including any changes that were made when the queue manager was inactive.

There are circumstances under which the refresh is not performed, for example:
- If a copy of the queue is open, a refresh that would change the usage of the queue fails.
- If a copy of a queue has messages on it, a refresh that would delete that queue fails.

In these circumstances, the refresh is not performed on that copy, but is performed on the copies on all other queue managers. Check for and correct any problems with copy objects after adding, changing, or deleting a group object, and at queue manager or channel initiator restart.

Managing the coupling facility:

Use this topic to understand how to add or remove coupling facility (CF) structures.

This section describes the following tasks:
- "Adding a coupling facility structure"
- "Removing a coupling facility structure"

Adding a coupling facility structure

There are no IBM MQ actions required when you add a coupling facility structure. The information about setting up the coupling facility in Task 10: Set up the coupling facility describes the rules for naming coupling facility structures, and how to define structures in the CFRM policy data set.

Removing a coupling facility structure

To remove a coupling facility structure, follow this procedure:
- Use the following command to get a list of all the queues using the coupling facility structure that you want to delete:

```
DISPLAY QUEUE(*) QSGDISP(SHARED) CFSTRUCT(structure-name)
```

- Delete all the queues that use the structure.
- Stop and restart each queue manager in the queue-sharing group in turn to disconnect IBM MQ and Db2 from the structure and delete information about it. (You do not need to stop all the queue managers at once; just one at a time.)
- Remove the structure definition from your CFRM policy data set and run the IXCMIAPU utility. (This is the reverse of customization task 10 (set up the coupling facility) described in Task 10: Set up the coupling facility.)

Recovery and restart

Use this topic to understand the recovery and restart mechanisms used by IBM MQ.

Restarting IBM MQ

After a queue manager terminates there are different restart procedures needed depending on how the queue manager terminated. Use this topic to understand the different restart procedures that you can use.

This topic contains information about how to restart your queue manager in the following circumstances:

- "Restarting after a normal shutdown"
- "Restarting after an abnormal termination"
- "Restarting if you have lost your page sets"
- "Restarting if you have lost your log data sets"
- Restarting if you have lost your CF structures

Restarting after a normal shutdown

If the queue manager was stopped with the STOP QMGR command, the system finishes its work in an orderly way and takes a termination checkpoint before stopping. When you restart the queue manager, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

To restart the queue manager, issue the START QMGR command as described in "Starting and stopping a queue manager" on page 268.

Restarting after an abnormal termination

IBM MQ automatically detects whether restart follows a normal shutdown or an abnormal termination.

Starting the queue manager after it has terminated abnormally is different from starting it after the STOP QMGR command has been issued. If the queue manager terminates abnormally, it terminates without being able to finish its work or take a termination checkpoint.

To restart the queue manager, issue the START QMGR command as described in "Starting and stopping a queue manager" on page 268. When you restart a queue manager after an abnormal termination, it refreshes its knowledge of its status at termination using information in the log, and notifies you of the status of various tasks.

Normally, the restart process resolves all inconsistent states. But, in some cases, you must take specific steps to resolve inconsistencies. This is described in "Recovering units of work manually" on page 354.

Restarting if you have lost your page sets

If you have lost your page sets, you need to restore them from your backup copies before you can restart the queue manager. This is described in "How to back up and recover page sets" on page 323.

The queue manager might take a long time to restart under these circumstances because of the length of time needed for media recovery.

Restarting if you have lost your log data sets

If, after stopping a queue manager (using the STOP QMGR command), both copies of the log are lost or damaged, you can restart the queue manager providing you have a consistent set of page sets (produced using Method 1: Full backup).

Follow this procedure:

1. Define new page sets to correspond to each existing page set in your queue manager. See Task 15: Define your page sets for information about page set definition.

   Ensure that each new page set is larger than the corresponding source page set.
2. Use the FORMAT function of CSQUTIL to format the destination page set. See Formatting page sets for more details.
3. Use the RESETPAGE function of CSQUTIL to copy the existing page sets or reset them in place, and reset the log RBA in each page. See Copying a page set and resetting the log for more information about this function.
4. Redefine your queue manager log data sets and BSDS using CSQJU003 (see The change log inventory utility).
5. Restart the queue manager using the new page sets. To do this, you do one of the following:
   - Change the queue manager started task procedure to reference the new page sets. See Task 6: Create procedures for the IBM MQ queue manager for more information.
   - Use Access Method Services to delete the old page sets and then rename the new page sets, giving them the same names as the old page sets.

Attention: Before you delete any IBM MQ page set, ensure that you have made the required backup copies.

If the queue manager is a member of a queue-sharing group, GROUP and SHARED object definitions are not normally affected by lost or damaged logs. However, if any shared-queue messages are involved in a unit of work that was covered by the lost or damaged logs, the effect on such uncommitted messages is unpredictable.

Note: If logs are damaged and the queue manager is a member of a queue-sharing group, the ability to recover shared persistent messages might be lost. Issue a BACKUP CFSTRUCT command immediately on another active queue manager in the queue-sharing group for all CF structures with the RECOVER(YES) attribute.

Restarting if you have lost your CF structures

You do not need to restart if you lose your CF structures, because the queue manager does not terminate.

Recovering a queue-sharing group at the alternative site:

If you need to recover a queue-sharing group, you must understand the steps needed to prepare the recovery, and the steps needed to recover the queue managers in the queue-sharing group.

Before you can recover the queue-sharing group, you need to prepare the environment:

1. If you have old information in your coupling faciltiy from practice startups when you installed the queue-sharing group, you need to clean this out first (if you do not have old information in the coupling faciltiy, you can omit this step:
   a. Enter the following z/OS command to display the CF structures for this queue-sharing group:

```
D XCF,STRUCTURE,STRNAME= qsgname*
```

b. For all structures that start with the queue-sharing group name, use the z/OS command SETXCF FORCE CONNECTION to force the connection off those structures:

```
SETXCF FORCE,CONNECTION,STRNAME= strname,CONNAME=ALL
```

c. Delete all the CF structures using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME= strname
```

2. Restore Db2 systems and data-sharing groups.
3. Recover the CSQ.ADMIN_B_STRBACKUP table so that it contains information about the most recent structure backups taken at the prime site.

   Note: It is important that the STRBACKUP table contains the most recent structure backup information. Older structure backup information might require data sets that you have discarded as a result of the information given by a recent DISPLAY USAGE TYPE(DATASET) command, which would mean that your recovered CF structure would not contain accurate information.
4. Run the ADD QMGR command of the CSQ5PQSG utility for every queue manager in the queue-sharing group. This will restore the XCF group entry for each queue manager.

   When you run the utility in this scenario, the following messages are normal:

```
CSQU566I Unable to get attributes for admin structure, CF not found
or not allocated
CSQU546E Unable to add QMGR <queue-manager-name> entry,
already exists in Db2 table CSQ.ADMIN_B_QMGR
CSQU148I CSQ5PQSG Utility completed, return code=4
```

To recover the queue managers in the queue-sharing group:

1. Define new page set data sets and load them with the data in the copies of the page sets from the primary site.
2. Define new active log data sets.
3. Define a new BSDS data set and use Access Method Services REPRO to copy the *most recent* archived BSDS into it.
4. Use the print log map utility CSQJU004 to print information from this most recent BSDS. At the time this BSDS was archived, the most recent archived log you have would have just been truncated as an active log, and does not appear as an archived log. Record the STARTRBA, STARTLRSN, ENDRBA, and ENDLRSN values of this log.
5. Use the change log inventory utility, CSQJU003, to register this latest archive log data set in the BSDS that you have just restored, using the values recorded in Step 4.
6. Use the DELETE option of CSQJU003 to remove all active log information from the BSDS.
7. Use the NEWLOG option of CSQJU003 to add active logs to the BSDS, do not specify STARTRBA or ENDRBA.
8. Calculate the `recoverylrsn` for the queue-sharing group. The `recoverylrsn` is the lowest of the ENDLRSNs across all queue managers in the queue-sharing group (as recorded in Step 4), minus 1. For example, if there are two queue managers in the queue-sharing group, and the ENDLRSN for one of them is B713 3C72 22C5, and for the other is B713 3D45 2123, the `recoverylrsn` is B713 3C72 22C4.
9. Use CSQJU003 to add a restart control record to the BSDS. Specify:

```
CRESTART CREATE,ENDLRSN= recoverylrsn
```

where *recoverylrsn* is the value you recorded in Step 8 on page 339.

The BSDS now describes all active logs as being empty, all the archived logs you have available, and no checkpoints beyond the end of your logs.

You must add the CRESTART record to the BSDS for each queue manager within the queue-sharing group.

10. Restart each queue manager in the queue-sharing group with the usual START QMGR command. During initialization, an operator reply message such as the following is issued:

```
CSQJ245D +CSQ1 RESTART CONTROL INDICATES TRUNCATION AT RBA highrba.
REPLY Y TO CONTINUE, N TO CANCEL
```

Reply `Y` to start the queue manager. The queue manager starts, and recovers data up to ENDRBA specified in the CRESTART statement.

At IBM MQ Version 7.0.1 and later, the first queue manager started can rebuild the admin structure partitions for other members of the queue sharing group as well as its own, and it is no longer necessary to restart each queue manager in the queue sharing group at this stage.

11. When the admin structure data for all queue managers has been rebuilt, issue a RECOVER CFSTRUCT command for each CF application structure.

If you issue the RECOVER CFSTRUCT command for all structures on a single queue manager, the log merge process is only performed once, so is quicker than issuing the command on a different queue manager for each CF structure, where each queue manager has to perform the log merge step.

When conditional restart processing is used in a queue sharing group, IBM WebSphere MQ Version 7.0.1 and later queue managers, performing peer admin rebuild, check that peers BSDS contain the same CRESTART LRSN as their own. This is to ensure the integrity of the rebuilt admin structure. It is therefore important to restart other peers in the QSG, so they can process their own CRESTART information, before the next unconditional restart of any member of the group.

Reinitializing a queue manager:

If the queue manager has terminated abnormally you might not be able to restart it. This could be because your page sets or logs have been lost, truncated, or corrupted. If this has happened, you might have to reinitialize the queue manager (perform a cold start).

Attention

Only perform a cold start if you cannot restart the queue manager any other way. Performing a cold start enables you to recover your queue manager and your object definitions; you will **not** be able to recover your message data. Check that none of the other restart scenarios described in this topic work for you before you do this.

When you have restarted, all your IBM MQ objects are defined and available for use, but there is no message data.

Note: Do not reinitialize a queue manager while it is part of a cluster. You must first remove the queue manager from the cluster (using RESET CLUSTER commands on the other queue managers in the cluster), then reinitialize it, and finally reintroduce it to the cluster as a new queue manager.

This is because during reinitialization, the queue manager identifier (QMID) is changed, so any cluster object with the old queue manager identifier must be removed from the cluster.

For further information see the following sections:
- Reinitializing a queue manager that is not in a queue-sharing group
- Reinitializing queue managers in a queue-sharing group

Reinitializing a queue manager that is not in a queue-sharing group

To reinitialize a queue manager, follow this procedure:
1. Prepare the object definition statements that to be used when you restart the queue manager. To do this, either:
   - If page set zero is available, use the CSQUTIL SDEFS function (see Producing a list of IBM MQ define commands). You must get definitions for all object types (authentication information objects, CF structures, channels, namelists, processes, queues, and storage classes).
   - If page set zero is not available, use the definitions from the last time you backed up your object definitions.
2. Redefine your queue manager data sets (do not do this until you have completed step 1).

   See creating the bootstrap and log data sets and defining your page sets for more information.
3. Restart the queue manager using the newly defined and initialized log data sets, BSDS, and page sets. Use the object definition input statements that you created in step 1 as input in the CSQINP2 initialization input data set.

Reinitializing queue managers in a queue-sharing group

In a queue-sharing group, reinitializing a queue manager is more complex. It might be necessary to reinitialize one or more queue managers because of page set or log problems, but there might also be problems with Db2 or the coupling facility to deal with. Because of this, there are a number of alternatives:

Cold start
Reinitializing the entire queue-sharing group involves forcing all the coupling facilities structures, clearing all object definitions for the queue-sharing group from Db2, deleting or redefining the logs and BSDS, and formatting page sets for all the queue managers in the queue-sharing group.

Shared definitions retained
Delete or redefine the logs and BSDS, format page sets for all queue managers in the queue-sharing group, and force all the coupling facilities structures. On restart, all messages will have been deleted. The queue managers re-create COPY objects that correspond to GROUP objects that still exist in the Db2 database. Any shared queues still exist and can be used.

Single queue manager reinitialized
Delete or redefine the logs and BSDS, and format page sets for the single queue manager (this deletes all its private objects and messages). On restart, the queue manager re-creates COPY objects that correspond to GROUP objects that still exist in the Db2 database. Any shared queues still exist, as do the messages on them, and can be used.

Point in time recovery of a queue-sharing group
This is the alternative site disaster recovery scenario.

Shared objects are recovered to the point in time achieved by Db2 recovery (described in A Db2 system fails). Each queue manager can be recovered to a point in time achievable from the backup copies available at the alternative site.

Persistent messages can be used in queue-sharing groups, and can be recovered using the MQSC RECOVER CFSTRUCT command. Note that this command recovers to the time of failure.

However, there is no recovery of nonpersistent shared queue messages; they are lost unless you have made backup copies independently using the COPY function of the CSQUTIL utility program.

It is not necessary to try to restore each queue manager to the same point in time because there are no interdependencies between the local objects on different queue managers (which are what is actually being recovered), and the queue manager resynchronization with Db2 on restart creates or deletes COPY objects as necessary on a queue manager by queue manager basis.

Alternative site recovery

z/OS

You can recover a single queue manager or a queue-sharing group, or consider disk mirroring.

See the following sections for more details:
- Recovering a single queue manager at an alternative site
- Recovering a queue-sharing group.
  - CF structure media recovery
  - Backing up the queue-sharing group at the prime site
  - Recovering a queue-sharing group at the alternative site
- Using disk mirroring

Recovering a single queue manager at an alternative site

If a total loss of an IBM MQ computing center occurs, you can recover on another queue manager or queue-sharing group at a recovery site. (See "Recovering a queue-sharing group at the alternative site" on page 345 for the alternative site recovery procedure for a queue-sharing group.)

To recover on another queue manager at a recovery site, you must regularly back up the page sets and the logs. As with all data recovery operations, the objectives of disaster recovery are to lose as little data, workload processing (updates), and time, as possible.

At the recovery site:
- The recovery queue managers **must** have the same names as the lost queue managers.
- The system parameter module (for example, CSQZPARM) used on each recovery queue manager must contain the same parameters as the corresponding lost queue manager.

When you have done this, reestablish all your queue managers as described in the following procedure. This can be used to perform disaster recovery at the recovery site for a single queue manager. It assumes that all that is available are:
- Copies of the archive logs and BSDSs created by normal running at the primary site (the active logs will have been lost along with the queue manager at the primary site).
- Copies of the page sets from the queue manager at the primary site that are the same age or older than the most recent archive log copies available.

You can use dual logging for the active and archive logs, in which case you need to apply the BSDS updates to both copies:
1. Define new page set data sets and load them with the data in the copies of the page sets from the primary site.
2. Define new active log data sets.
3. Define a new BSDS data set and use Access Method Services REPRO to copy the *most recent* archived BSDS into it.

4. Use the print log map utility CSQJU004 to print information from this most recent BSDS. At the time this BSDS was archived, the most recent archived log you have would have just been truncated as an active log, and does not appear as an archived log. Record the STARTRBA and ENDRBA of this log.
5. Use the change log inventory utility, CSQJU003, to register this latest archive log data set in the BSDS that you have just restored, using the STARTRBA and ENDRBA recorded in Step 4.
6. Use the DELETE option of CSQJU003 to remove all active log information from the BSDS.
7. Use the NEWLOG option of CSQJU003 to add active logs to the BSDS, do not specify STARTRBA or ENDRBA.
8. Use CSQJU003 to add a restart control record to the BSDS. Specify `CRESTART CREATE,ENDRBA=highrba`, where `highrba` is the high RBA of the most recent archive log available (found in Step 4), plus 1.

   The BSDS now describes all active logs as being empty, all the archived logs you have available, and no checkpoints beyond the end of your logs.
9. Restart the queue manager with the START QMGR command. During initialization, an operator reply message such as the following is issued:

```
CSQJ245D +CSQ1 RESTART CONTROL INDICATES TRUNCATION AT RBA highrba.
REPLY Y TO CONTINUE, N TO CANCEL
```

   Type `Y` to start the queue manager. The queue manager starts, and recovers data up to ENDRBA specified in the CRESTART statement.

See Using the IBM MQ utilities for information about using CSQJU003 and CSQJU004.

Figure 50 shows sample input statements for CSQJU003 for steps 6, 7, and 8:

```
* Step 6
DELETE DSNAME=MQM2.LOGCOPY1.DS01
DELETE DSNAME=MQM2.LOGCOPY1.DS02
DELETE DSNAME=MQM2.LOGCOPY1.DS03
DELETE DSNAME=MQM2.LOGCOPY1.DS04
DELETE DSNAME=MQM2.LOGCOPY2.DS01
DELETE DSNAME=MQM2.LOGCOPY2.DS02
DELETE DSNAME=MQM2.LOGCOPY2.DS03
DELETE DSNAME=MQM2.LOGCOPY2.DS04

* Step 7
NEWLOG DSNAME=MQM2.LOGCOPY1.DS01,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY1.DS02,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY1.DS03,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY1.DS04,COPY1
NEWLOG DSNAME=MQM2.LOGCOPY2.DS01,COPY2
NEWLOG DSNAME=MQM2.LOGCOPY2.DS02,COPY2
NEWLOG DSNAME=MQM2.LOGCOPY2.DS03,COPY2
NEWLOG DSNAME=MQM2.LOGCOPY2.DS04,COPY2

* Step 8
CRESTART CREATE,ENDRBA=063000
```

Figure 50. Sample input statements for CSQJU003

The things you need to consider for restarting the channel initiator at the recovery site are like those faced when using ARM to restart the channel initiator on a different z/OS image. See"Using ARM in an IBM MQ network" on page 352for more information. Your recovery strategy should also cover recovery of the IBM MQ product libraries and the application programming environments that use IBM MQ (CICS , for example).

Other functions of the change log inventory utility (CSQJU003) can also be used in disaster recovery scenarios. The HIGHRBA function allows the update of the highest RBA written and highest RBA offloaded values within the bootstrap data set. The CHECKPT function allows the addition of new checkpoint queue records or the deletion of existing checkpoint queue records in the BSDS.

Attention: These functions might affect the integrity of your IBM MQ data. Only use them in disaster recovery scenarios under the guidance of IBM service personnel.

Fast copy techniques

If copies of all the page sets and logs are made while the queue manager is frozen, the copies will be a consistent set that can be used to restart the queue manager at an alternative site. They typically enable a much faster restart of the queue manager, as there is little media recovery to be performed.

Use the SUSPEND QMGR LOG command to freeze the queue manager. This command flushes buffer pools to the page sets, takes a checkpoint, and stops any further log write activity. Once log write activity has been suspended, the queue manager is effectively frozen until you issue a RESUME QMGR LOG command. While the queue manager is frozen, the page sets and logs can be copied.

By using copying tools such as FLASHCOPY or SNAPSHOT to rapidly copy the page sets and logs, the time during which the queue manager is frozen can be reduced to a minimum.

Within a queue-sharing group, however, the SUSPEND QMGR LOG command might not be such a good solution. To be effective, the copies of the logs must all contain the same point in time for recovery, which means that the SUSPEND QMGR LOG command must be issued on all queue managers within the queue-sharing group simultaneously, and therefore the entire queue-sharing group will be frozen for some time.

Recovering a queue-sharing group

In the event of a prime site disaster, you can restart a queue-sharing group at a remote site using backup data sets from the prime site. To recover a queue-sharing group you need to coordinate the recovery across all the queue managers in the queue-sharing group, and coordinate with other resources, primarily Db2. This section describes these tasks in detail.

- CF structure media recovery
- Backing up the queue-sharing group at the prime site
- Recovering a queue-sharing group at the alternative site

CF structure media recovery

Media recovery of a CF structure used to hold persistent messages on a shared queue, relies on having a backup of the media that can be forward recovered by the application of logged updates. Take backups of your CF structures periodically using the MQSC BACKUP CFSTRUCT command. All updates to shared queues (**MQGET** s and **MQPUT** s) are written on the log of the queue manager where the update is performed. To perform media recovery of a CF structure you must apply logged updates to that backup from the logs of **all** the queue managers that have used that CF structure. When you use the MQSC RECOVER CFSTRUCT command, IBM MQ automatically merges the logs from relevant queue managers, and applies the updates to the most recent backup.

The CF structure backup is written to the log of the queue manager that processed the BACKUP CFSTRUCT command, so there are no additional data sets to be collected and transported to the alternative site.

Backing up the queue-sharing group at the prime site

At the prime site you need to establish a consistent set of backups on a regular basis, which can be used in the event of a disaster to rebuild the queue-sharing group at an alternative site. For a single queue manager, recovery can be to an arbitrary point in time, typically the end of the logs available at the remote site. However, where persistent messages have been stored on a shared queue, the logs of all the queue managers in the queue-sharing group must be merged to recover shared queues, as any queue manager in the queue-sharing group might have performed updates (**MQPUT** s or **MQGET** s) on the queue.

For recovery of a queue-sharing group, you need to establish a point in time that is within the log range of the log data of all queue managers. However, as you can only **forward** recover media from the log, this point in time must be after the BACKUP CFSTRUCT command has been issued and after any page set backups have been performed. (Typically, the point in time for recovery might correspond to the end of a business day or week.)

The following diagram shows time lines for two queue managers in a queue-sharing group. For each queue manager, fuzzy backups of page sets are taken (see Method 2: Fuzzy backup). On queue manager A, a BACKUP CFSTRUCT command is issued. Subsequently, an ARCHIVE LOG command is issued on each queue manager to truncate the active log, and copy it to media offline from the queue manager, which can be transported to the alternative site. 'End of log' identifies the time at which the ARCHIVE LOG command was issued, and therefore marks the extent of log data typically available at the alternative site. The point in time for recovery must lie between the end of any page set or CF structure backups, and the earliest end of log available at the alternative site.

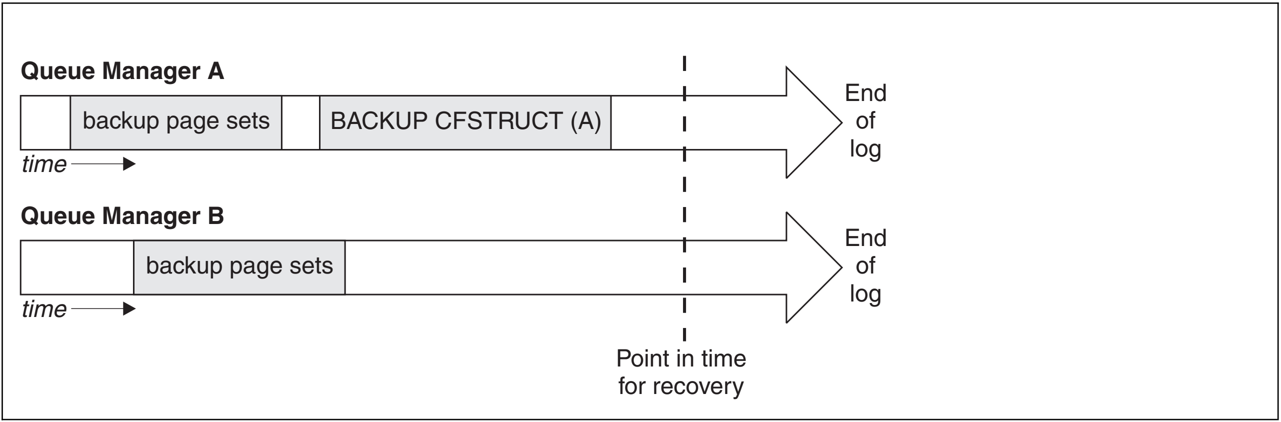


Figure 51. Point in time for recovery for 2 queue managers in a queue-sharing group

IBM MQ records information associated with the CF structure backups in a table in Db2. Depending on your requirements, you might want to coordinate the point in time for recovery of IBM MQ with that for Db2, or it might be sufficient to take a copy of the IBM MQ CSQ.ADMIN_B_STRBACKUP table after the BACKUP CFSTRUCT commands have finished.

To prepare for a recovery:

1. Create page set backups for each queue manager in the queue-sharing group.
2. Issue a BACKUP CFSTRUCT command for each CF structure with the RECOVER(YES) attribute. You can issue these commands from a single queue manager, or from different queue managers within the queue-sharing group to balance the workload.
3. Once all the backups have completed, issue an ARCHIVE LOG command to switch the active log and create copies of the logs and BSDSs of each queue manager in the queue-sharing group.
4. Transport the page set backups, the archived logs, the archived BSDS of all the queue managers in the queue-sharing group, and your chosen Db2 backup information, off-site.

Recovering a queue-sharing group at the alternative site

Before you can recover the queue-sharing group, you need to prepare the environment:

1. If you have old information in your coupling facility from practice startups when you installed the queue-sharing group, you need to clean this out first (if you do not have old information in the coupling facility, you can omit this step:
   a. Enter the following z/OS command to display the CF structures for this queue-sharing group:

```
D XCF,STRUCTURE,STRNAME= qsgname
```

   b. For all structures that start with the queue-sharing group name, use the z/OS command SETXCF FORCE CONNECTION to force the connection off those structures:

```
SETXCF FORCE,CONNECTION,STRNAME= strname,CONNAME=ALL
```

   c. Delete all the CF structures using the following command for each structure:

```
SETXCF FORCE,STRUCTURE,STRNAME= strname
```

2. Restore Db2 systems and data-sharing groups.
3. Recover the CSQ.ADMIN_B_STRBACKUP table so that it contains information about the most recent structure backups taken at the prime site.

   Note: It is important that the STRBACKUP table contains the most recent structure backup information. Older structure backup information might require data sets that you have discarded as a result of the information given by a recent DISPLAY USAGE TYPE(DATASET) command, which would mean that your recovered CF structure would not contain accurate information.
4. Run the ADD QMGR command of the CSQ5PQSG utility for every queue manager in the queue-sharing group. This will restore the XCF group entry for each queue manager.

To recover the queue managers in the queue-sharing group:

1. Define new page set data sets and load them with the data in the copies of the page sets from the primary site.
2. Define new active log data sets.
3. Define a new BSDS data set and use Access Method Services REPRO to copy the *most recent* archived BSDS into it.
4. Use the print log map utility CSQJU004 to print information from this most recent BSDS. At the time this BSDS was archived, the most recent archived log you have would have just been truncated as an active log, and does not appear as an archived log. Record the STARTRBA, STARTLRSN, ENDRBA, and ENDLRSN values of this log.
5. Use the change log inventory utility, CSQJU003, to register this latest archive log data set in the BSDS that you have just restored, using the values recorded in Step 4.
6. Use the DELETE option of CSQJU003 to remove all active log information from the BSDS.
7. Use the NEWLOG option of CSQJU003 to add active logs to the BSDS, do not specify STARTRBA or ENDRBA.
8. Calculate the `recoverylrsn` for the queue-sharing group. The `recoverylrsn` is the lowest of the ENDLRSNs across all queue managers in the queue-sharing group (as recorded in Step 4), minus 1. For example, if there are two queue managers in the queue-sharing group, and the ENDLRSN for one of them is B713 3C72 22C5, and for the other is B713 3D45 2123, the `recoverylrsn` is B713 3C72 22C4.

9. Use CSQJU003 to add a restart control record to the BSDS. Specify:

```
CRESTART CREATE,ENDLRSN= recoverylrsn
```

   where *recoverylrsn* is the value you recorded in Step 8 on page 346.

   The BSDS now describes all active logs as being empty, all the archived logs you have available, and no checkpoints beyond the end of your logs.

   You must add the CRESTART record to the BSDS for each queue manager within the queue-sharing group.

10. Restart each queue manager in the queue-sharing group with the START QMGR command. During initialization, an operator reply message such as the following is issued:

```
CSQJ245D +CSQ1 RESTART CONTROL INDICATES TRUNCATION AT RBA highrba.
REPLY Y TO CONTINUE, N TO CANCEL
```

   Reply Y to start the queue manager. The queue manager starts, and recovers data up to ENDRBA specified in the CRESTART statement.

   At IBM
   WebSphere MQ Version 7.0.1 and later, the first queue manager started can rebuild the admin structu

11. When the admin structure data for all queue managers has been rebuilt, issue a RECOVER CFSTRUCT command for each CF application structure.

   If you issue the RECOVER CFSTRUCT command for all structures on a single queue manager, the log merge process is only performed once, so is quicker than issuing the command on a different queue manager for each CF structure, where each queue manager has to perform the log merge step.

When conditional restart processing is used in a queue sharing group, IBM WebSphere MQ Version 7.0.1 and later queue managers, performing peer admin rebuild, check that peers BSDS contain the same CRESTART LRSN as their own. This is to ensure the integrity of the rebuilt admin structure. It is therefore important to restart other peers in the QSG, so they can process their own CRESTART information, before the next unconditional restart of any member of the group.

Using disk mirroring

Many installations now use disk mirroring technologies such as IBM Metro Mirror (formerly PPRC) to make synchronous copies of data sets at an alternative site. In such situations, many of the steps detailed become unnecessary as the IBM MQ page sets and logs at the alternative site are effectively identical to those at the prime site. Where such technologies are used, the steps to restart a queue sharing group at an alternative site may be summarized as:

- Clear IBM MQ CF structures at the alternative site. (These often contain residual information from any previous disaster recovery exercise).
- Restore Db2 systems and all tables in the database used by the IBM MQ queue sharing group.
- Restart queue managers. Before IBM WebSphere MQ Version 7.0.1, it is necessary to restart each queue manager defined in the queue sharing group as each queue manage recovers its own partition of the admin structure during queue manager restart. After each queue manager has been restarted, those not on their 'home' LPAR can be shut down again. For IBM WebSphere MQ Version 7.0.1 and later, the first queue manager started rebuilds the admin structure partitions for other members of the queue sharing group as well as its own, and it is no longer necessary to restart each queue manager in the queue sharing group.

- After the admin structure has been rebuilt, recover the application structures.

Reinitializing a queue manager

If the queue manager has terminated abnormally you might not be able to restart it. This could be because your page sets or logs have been lost, truncated, or corrupted. If this has happened, you might have to reinitialize the queue manager (perform a cold start).

Attention

Only perform a cold start if you cannot restart the queue manager any other way. Performing a cold start enables you to recover your queue manager and your object definitions; you will **not** be able to recover your message data. Check that none of the other restart scenarios described in this topic work for you before you do this.

When you have restarted, all your IBM MQ objects are defined and available for use, but there is no message data.

Note: Do not reinitialize a queue manager while it is part of a cluster. You must first remove the queue manager from the cluster (using RESET CLUSTER commands on the other queue managers in the cluster), then reinitialize it, and finally reintroduce it to the cluster as a new queue manager.

This is because during reinitialization, the queue manager identifier (QMID) is changed, so any cluster object with the old queue manager identifier must be removed from the cluster.

For further information see the following sections:
- Reinitializing a queue manager that is not in a queue-sharing group
- Reinitializing queue managers in a queue-sharing group

Reinitializing a queue manager that is not in a queue-sharing group

To reinitialize a queue manager, follow this procedure:
1. Prepare the object definition statements that to be used when you restart the queue manager. To do this, either:
   - If page set zero is available, use the CSQUTIL SDEFS function (see Producing a list of IBM MQ define commands). You must get definitions for all object types (authentication information objects, CF structures, channels, namelists, processes, queues, and storage classes).
   - If page set zero is not available, use the definitions from the last time you backed up your object definitions.
2. Redefine your queue manager data sets (do not do this until you have completed step 1 on page 341).

   See creating the bootstrap and log data sets and defining your page sets for more information.
3. Restart the queue manager using the newly defined and initialized log data sets, BSDS, and page sets. Use the object definition input statements that you created in step 1 on page 341 as input in the CSQINP2 initialization input data set.

Reinitializing queue managers in a queue-sharing group

In a queue-sharing group, reinitializing a queue manager is more complex. It might be necessary to reinitialize one or more queue managers because of page set or log problems, but there might also be problems with Db2 or the coupling facility to deal with. Because of this, there are a number of alternatives:

Cold start
Reinitializing the entire queue-sharing group involves forcing all the coupling facilities structures,

clearing all object definitions for the queue-sharing group from Db2, deleting or redefining the logs and BSDS, and formatting page sets for all the queue managers in the queue-sharing group.

Shared definitions retained
Delete or redefine the logs and BSDS, format page sets for all queue managers in the queue-sharing group, and force all the coupling facilities structures. On restart, all messages will have been deleted. The queue managers re-create COPY objects that correspond to GROUP objects that still exist in the Db2 database. Any shared queues still exist and can be used.

Single queue manager reinitialized
Delete or redefine the logs and BSDS, and format page sets for the single queue manager (this deletes all its private objects and messages). On restart, the queue manager re-creates COPY objects that correspond to GROUP objects that still exist in the Db2 database. Any shared queues still exist, as do the messages on them, and can be used.

Point in time recovery of a queue-sharing group
This is the alternative site disaster recovery scenario.

Shared objects are recovered to the point in time achieved by Db2 recovery (described in A Db2 system fails). Each queue manager can be recovered to a point in time achievable from the backup copies available at the alternative site.

Persistent messages can be used in queue-sharing groups, and can be recovered using the MQSC RECOVER CFSTRUCT command. Note that this command recovers to the time of failure. However, there is no recovery of nonpersistent shared queue messages; they are lost unless you have made backup copies independently using the COPY function of the CSQUTIL utility program.

It is not necessary to try to restore each queue manager to the same point in time because there are no interdependencies between the local objects on different queue managers (which are what is actually being recovered), and the queue manager resynchronization with Db2 on restart creates or deletes COPY objects as necessary on a queue manager by queue manager basis.

Using the z/OS Automatic Restart Manager (ARM)

Use this topic to understand how you can use ARM to automatically restart your queue managers.

This section contains information about the following topics:
- "What is the ARM?"
- "ARM policies" on page 350
- "Using ARM in an IBM MQ network" on page 352

What is the ARM?

The z/OS Automatic Restart Manager (ARM) is a z/OS recovery function that can improve the availability of your queue managers. When a job or task fails, or the system on which it is running fails, ARM can restart the job or task without operator intervention.

If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS, and hence a whole group of related subsystems and applications have failed, ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a *cross-system restart*.

Restart the channel initiator by ARM only in exceptional circumstances. If the queue manager is restarted by ARM, restart the channel initiator from the CSQINP2 initialization data set (see "Using ARM in an IBM MQ network" on page 352).

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure. The network implications of IBM MQ ARM restart on a different z/OS image are described in "Using ARM in an IBM MQ network" on page 352.

To enable automatic restart:
- Set up an ARM couple data set.
- Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.
- Start the ARM policy.

Also, IBM MQ must register with ARM at startup (this happens automatically).

Note: If you want to restart queue managers in different z/OS images automatically, you must define every queue manager as a subsystem in each z/OS image on which that queue manager might be restarted, with a sysplex wide unique four character subsystem name.

ARM couple data sets

Ensure that you define the couple data sets required for ARM, and that they are online and active before you start any queue manager for which you want ARM support. IBM MQ automatic ARM registration fails if the couple data sets are not available at queue manager startup. In this situation, IBM MQ assumes that the absence of the couple data set means that you do not want ARM support, and initialization continues.

See the *z/OS MVS Setting up a Sysplex* manual for information about ARM couple data sets.

ARM policies:

The Automatic Restart Manager policies are user-defined rules that control ARM functions that can control any restarts of a queue manager.

ARM functions are controlled by a user-defined *ARM policy*. Each z/OS image running a queue manager instance that is to be restarted by ARM must be connected to an ARM couple data set with an active ARM policy.

IBM provides a default ARM policy. You can define new policies, or override the policy defaults by using the *administrative data utility* (IXCMIAPU) provided with z/OS. The *z/OS MVS Setting up a Sysplex* manual describes this utility, and includes full details of how to define an ARM policy.

Figure 52 shows an example of an ARM policy. This sample policy restarts any queue manager within a sysplex, if either the queue manager failed, or a whole system failed.

```
//IXCMIAPU EXEC PGM=IXCMIAPU,REGION=2M
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
DATA TYPE(ARM)
DEFINE POLICY NAME(ARMPOL1) REPLACE(YES)
RESTART_GROUP(DEFAULT)
ELEMENT(*)
RESTART_ATTEMPTS(0)  /* Jobs not to be restarted by ARM  */
RESTART_GROUP(GROUP1)
ELEMENT(SYSMQMGRMQ*)  /* These jobs to be restarted by ARM */
/*
```

Figure 52. Sample ARM policy

For more information see:
- Defining an ARM policy

- Activating an ARM policy
- Registering with ARM

Defining an ARM policy

Set up your ARM policy as follows:
- Define RESTART_GROUPs for each queue manager instance that also contain any CICS or IMS subsystems that connect to that queue manager instance. If you use a subsystem naming convention, you might be able to use the '?' and '*' wild-card characters in your element names to define RESTART_GROUPs with minimum definition effort.
- Specify TERMTYPE(ELEMTERM) for your channel initiators to indicate that they will be restarted only if the channel initiator has failed and the z/OS image has not failed.
- Specify TERMTYPE(ALLTERM) for your queue managers to indicate that they will be restarted if either the queue manager has failed or the z/OS image has failed.
- Specify RESTART_METHOD(BOTH, PERSIST) for both queue managers and channel initiators. This tells ARM to restart using the JCL it saved (after resolution of system symbols) during the last startup. It tells ARM to do this irrespective of whether the individual element failed, or the z/OS image failed.
- Accept the default values for all the other ARM policy options.

Activating an ARM policy

To start your automatic restart management policy, issue the following z/OS command:

```
SETXCF START,POLICY,TYPE=ARM,POLNAME= mypol
```

When the policy is started, all systems connected to the ARM couple data set use the same active policy.

Use the SETXCF STOP command to disable automatic restarts.

Registering with ARM

IBM MQ registers automatically as an *ARM element* during queue manager startup (subject to ARM availability). It deregisters during its shutdown phase, unless requested not to.

At startup, the queue manager determines whether ARM is available. If it is, IBM MQ registers using the name SYSMQMGR *ssid*, where *ssid* is the four character queue manager name, and SYSMQMGR is the element type.

The STOP QMGR MODE(QUIESCE) and STOP QMGR MODE(FORCE) commands deregister the queue manager from ARM (if it was registered with ARM at startup). This prevents ARM restarting this queue manager. The STOP QMGR MODE(RESTART) command does not deregister the queue manager from ARM, so it is eligible for immediate automatic restart.

Each channel initiator address space determines whether ARM is available, and if so registers with the element name SYSMQCH *ssid*, where *ssid* is the queue manager name, and SYSMQCH is the element type.

The channel initiator is always deregistered from ARM when it stops normally, and remains registered only if it ends abnormally. The channel initiator is always deregistered if the queue manager fails.

Using ARM in an IBM MQ network:

You can set up your queue manager so that the channel initiators and associated listeners are started automatically when the queue manager is restarted.

To ensure fully automatic queue manager restart on the same z/OS image for both LU 6.2 and TCP/IP communication protocols:

- Start your listeners automatically by adding the appropriate START LISTENER command to the CSQINPX data set.
- Start your channel initiator automatically by adding the appropriate START CHINIT command to the CSQINP2 data set.

For restarting a queue manager with TCP/IP or LU6.2, see

- "Restarting on a different z/OS image with TCP/IP"
- "Restarting on a different z/OS image with LU 6.2" on page 353

See Task 13: Customize the initialization input data sets for information about the CSQINP2 and CSQINPX data sets.

Restarting on a different z/OS image with TCP/IP

If you are using TCP/IP as your communication protocol, and you are using virtual IP addresses, you can configure these to recover on other z/OS images, allowing channels connecting to that queue manager to reconnect without any changes. Otherwise, you can reallocate a TCP/IP address after moving a queue manager to a different z/OS image only if you are using clusters or if you are connecting to a queue-sharing group using a WLM dynamic Domain Name System (DNS) logical group name.

- When using clustering
- When connecting to a queue-sharing group

When using clustering

z/OS ARM responds to a system failure by restarting the queue manager on a different z/OS image in the same sysplex; this system has a different TCP/IP address to the original z/OS image. The following explains how you can use IBM MQ clusters to reassign a queue manager's TCP/IP address after it has been moved by ARM restart to a different z/OS image.

When a client queue manager detects the queue manager failure (as a channel failure), it responds by reallocating suitable messages on its cluster transmission queue to a different server queue manager that hosts a different instance of the target cluster queue. However, it cannot reallocate messages that are bound to the original server by affinity constraints, or messages that are in doubt because the server queue manager failed during end-of-batch processing. To process these messages, do the following:

1. Allocate a different cluster-receiver channel name and a different TCP/IP port to each z/OS queue manager. Each queue manager needs a different port so that two systems can share a single TCP/IP stack on a z/OS image. One of these is the queue manager originally running on that z/OS image, and the other is the queue manager that ARM will restart on that z/OS image following a system failure. Configure each port on each z/OS image, so that ARM can restart any queue manager on any z/OS image.
2. Create a different channel initiator command input file (CSQINPX) for each queue manager and z/OS image combination, to be referenced during channel initiator startup.

   Each CSQINPX file must include a START LISTENER PORT(port) command specific to that queue manager, and an ALTER CHANNEL command for a cluster-receiver channel specific to that queue manager and z/OS image combination. The ALTER CHANNEL command needs to set the connection name to the TCP/IP name of the z/OS image on which it is restarted. It must include the port number specific to the restarted queue manager as part of the connection name.

The start-up JCL of each queue manager can have a fixed data set name for this CSQINPX file, and each z/OS image must have a different version of each CSQINPX file on a non-shared DASD volume.

If an ARM restart occurs, IBM MQ advertises the changed channel definition to the cluster repository, which in turn publishes it to all the client queue managers that have expressed an interest in the server queue manager.

The client queue manager treats the server queue manager failure as a channel failure, and tries to restart the failed channel. When the client queue manager learns the new server connection-name, the channel restart reconnects the client queue manager to the restarted server queue manager. The client queue manager can then resynchronize its messages, resolve any in-doubt messages on the client queue manager's transmission queue, and normal processing can continue.

When connecting to a queue-sharing group

When connecting to a queue-sharing group through a TCP/IP dynamic Domain Name System (DNS) logical group name, the connection name in your channel definition specifies the logical group name of your queue-sharing group, not the host name or IP address of a physical machine. When this channel starts, it connects to the dynamic DNS and is then connected to one of the queue managers in the queue-sharing group. This process is explained in Setting up communication for IBM MQ for z/OS using queue-sharing groups.

In the unlikely event of an image failure, one of the following occurs:

- The queue managers on the failing image de-register from the dynamic DNS running on your sysplex. The channel responds to the connection failure by entering RETRYING state and then connects to the dynamic DNS running on the sysplex. The dynamic DNS allocates the inbound request to one of the remaining members of the queue-sharing group that is still running on the remaining images.
- If no other queue manager in the queue-sharing group is active and ARM restarts the queue manager and channel initiator on a different image, the group listener registers with dynamic DNS from this new image. This means that the logical group name (from the connection name field of the channel) connects to the dynamic DNS and is then connected to the same queue manager, now running on a different image. No change was required to the channel definition.

For this type of recovery to occur, the following points must be noted:

- On z/OS, the dynamic DNS runs on one of the z/OS images in the sysplex. If this image were to fail, the dynamic DNS needs to be configured so that there is a secondary name server active in the sysplex, acting as an alternative to the primary name server. Information about primary and secondary dynamic DNS servers can be found in the *OS/390® SecureWay CS IP Configuration* manual.
- The TCP/IP group listener might have been started on a particular IP address that might not be available on this z/OS image. If so, the listener might need to be started on a different IP address on the new image. If you are using virtual IP addresses, you can configure these to recover on other z/OS images so that no change to the START LISTENER command is required.

Restarting on a different z/OS image with LU 6.2

If you use only LU 6.2 communication protocols, carry out the following procedure to enable network reconnect after automatic restart of a queue manager on a different z/OS image within the sysplex:

- Define each queue manager within the sysplex with a unique subsystem name.

- Define each channel initiator within the sysplex with a unique LUNAME. This is specified in both the queue manager attributes and in the START LISTENER command.

  Note: The LUNAME names an entry in the APPC side table, which in turn maps this to the actual LUNAME.
- Set up a shared APPC side table, which is referenced by each z/OS image within the sysplex. This should contain an entry for each channel initiator's LUNAME. See the *MVS Planning: APPC/MVS Management* manual for information about this.
- Set up an APPCPM *xx* member of SYS1.PARMLIB for each channel initiator within the sysplex to contain an LUADD to activate the APPC side table entry for that channel initiator. These members should be shared by each z/OS image. The appropriate SYS1.PARMLIB member is activated by a z/OS command SET APPC= *xx*, which is issued automatically during ARM restart of the queue manager (and its channel initiator) on a different z/OS image, as described in the following text.
- Use the LU62ARM queue manager attribute to specify the *xx* suffix of this SYS1.PARMLIB member for each channel initiator. This causes the channel initiator to issue the required z/OS command SET APPC= *xx* to activate its LUNAME.

Define your ARM policy so that it restarts the channel initiator only if it fails while its z/OS image stays up; the user ID associated with the XCFAS address space must be authorized to issue the IBM MQ command START CHINIT. Do not restart the channel initiator automatically if its z/OS image also fails, instead use commands in the CSQINP2 and CSQINPX data sets to start the channel initiator and listeners.

Recovering units of work manually

You can manually recover units of work CICS, IMS, RRS, or other queue managers in a queue-sharing group. You can use queue manager commands to display the status of the units of work associated with each connection to the queue manager.

This topic contains information about the following subjects:
- “Displaying connections and threads”
- “Recovering CICS units of recovery manually” on page 355
- “Recovering IMS units of recovery manually” on page 358
- “Recovering RRS units of recovery manually” on page 361
- “Recovering units of recovery on another queue manager in the queue-sharing group” on page 361

Displaying connections and threads

You can use the DISPLAY CONN command to get information about connections to queue managers and their associated units of work. You can display active units of work to see what is currently happening, or to see what needs to be terminated to allow the queue manager to shut down, and you can display unresolved units of work to help with recovery.

Active units of work

To display only active units of work, use

```
DISPLAY CONN(*) WHERE(UOWSTATE EQ ACTIVE)
```

Unresolved units of work

An unresolved unit of work, also known as an "in-doubt thread", is one that is in the second pass of the two-phase commit operation. Resources are held in IBM MQ on its behalf. To display unresolved units of work, use

```
DISPLAY CONN(*) WHERE(UOWSTATE EQ UNRESOLVED)
```

External intervention is needed to resolve the status of unresolved units of work. This might only involve starting the recovery coordinator (CICS, IMS, or RRS) or might involve more, as described in the following sections.

Recovering CICS units of recovery manually:

Use this topic to understand what happens when the CICS adapter restarts, and then explains how to deal with any unresolved units of recovery that arise.

What happens when the CICS adapter restarts

Whenever a connection is broken, the adapter has to go through a *restart phase* during the *reconnect process*. The restart phase resynchronizes resources. Resynchronization between CICS and IBM MQ enables in-doubt units of work to be identified and resolved.

Resynchronization can be caused by:
- An explicit request from the distributed queuing component
- An implicit request when a connection is made to IBM MQ

If the resynchronization is caused by connecting to IBM MQ, the sequence of events is:
1. The connection process retrieves a list of in-doubt units of work (UOW) IDs from IBM MQ.
2. The UOW IDs are displayed on the console in CSQC313I messages.
3. The UOW IDs are passed to CICS.
4. CICS initiates a resynchronization task (CRSY) for each in-doubt UOW ID.
5. The result of the task for each in-doubt UOW is displayed on the console.

You need to check the messages that are displayed during the connect process:

CSQC313I
Shows that a UOW is in doubt.

CSQC400I
Identifies the UOW and is followed by one of these messages:
- CSQC402I or CSQC403I shows that the UOW was resolved successfully (committed or backed out).
- CSQC404E, CSQC405E, CSQC406E, or CSQC407E shows that the UOW was not resolved.

CSQC409I
Shows that all UOWs were resolved successfully.

CSQC408I
Shows that not all UOWs were resolved successfully.

CSQC314I
Warns that UOW IDs highlighted with a * are not resolved automatically. These UOWs must be resolved explicitly by the distributed queuing component when it is restarted.

Figure 53 on page 356 shows an example set of restart messages displayed on the z/OS console.

```
CSQ9022I +CSQ1 CSQYASCP ' START QMGR' NORMAL COMPLETION
+CSQC323I VICIC1 CSQCQCON CONNECT received from TERMID=PB62 TRANID=CKCN
+CSQC303I VICIC1 CSQCCON CSQCSERV loaded. Entry point is 850E8918
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6F0E2178D25 is in doubt
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6F055B2AC25 is in doubt
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6EFFD60D425 is in doubt
+CSQC313I VICIC1 CSQCCON UOWID=VICIC1.A6E5A6F07AB56D22 is in doubt
+CSQC307I VICIC1 CSQCCON Successful connection to subsystem VC2
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BAD18) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BAA10) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BA708) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CAE88) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CAB80) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA878) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA570) connect
successful
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA268) connect
successful
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F0E2178D25
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F055B2AC25
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F07AB56D22
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6EFFD60D425
+CSQC409I VICIC1 CSQCTRUE Resynchronization completed successfully
```

Figure 53. Example restart messages

The total number of CSQC313I messages should equal the total number of CSQC402I plus CSQC403I messages. If the totals are not equal, there are UOWs that the connection process cannot resolve. Those UOWs that cannot be resolved are caused by problems with CICS (for example, a cold start) or with IBM MQ, or by distributing queuing. When these problems have been fixed, you can initiate another resynchronization by disconnecting and then reconnecting.

Alternatively, you can resolve each outstanding UOW yourself using the RESOLVE INDOUBT command and the UOW ID shown in message CSQC400I. You must then initiate a disconnect and a connect to clean up the *unit of recovery descriptors* in CICS. You need to know the correct outcome of the UOW to resolve UOWs manually.

All messages that are associated with unresolved UOWs are locked by IBM MQ and no Batch, TSO, or CICS task can access them.

If CICS fails and an emergency restart is necessary, *do not* vary the GENERIC APPLID of the CICS system. If you do and then reconnect to IBM MQ, data integrity with IBM MQ cannot be guaranteed. This is because IBM MQ treats the new instance of CICS as a different CICS (because the APPLID is different). In-doubt resolution is then based on the wrong CICS log.

How to resolve CICS units of recovery manually

If the adapter ends abnormally, CICS and IBM MQ build in-doubt lists either dynamically or during restart, depending on which subsystem caused the abend.

Note: If you use the DFH$INDB sample program to show units of work, you might find that it does not always show IBM MQ UOWs correctly.

When CICS connects to IBM MQ, there might be one or more units of recovery that have not been resolved.

One of the following messages is sent to the console:
- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E
- CSQC408I

For details of what these messages mean, see the CICS adapter and Bridge messages messages.

CICS retains details of units of recovery that were not resolved during connection startup. An entry is purged when it no longer appears on the list presented by IBM MQ.

Any units of recovery that CICS cannot resolve must be resolved manually using IBM MQ commands. This manual procedure is rarely used within an installation, because it is required only where operational errors or software problems have prevented automatic resolution. *Any inconsistencies found during in-doubt resolution must be investigated.*

To resolve the units of recovery:
1. Obtain a list of the units of recovery from IBM MQ using the following command:

```
+CSQ1  DISPLAY CONN( * ) WHERE(UOWSTATE EQ UNRESOLVED)
```

   You receive the following message:

```
CSQM201I +CSQ1 CSQMDRTC DISPLAY CONN DETAILS
CONN(BC85772CBE3E0001)
EXTCONN(C3E2D8C3C7D9F0F94040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2005-02-04)
UOWLOGTI(10.17.44)
UOWSTDA(2005-02-04)
UOWSTTI(10.17.44)
UOWSTATE(UNRESOLVED)
NID(IYRCSQ1 .BC8571519B60222D)
EXTURID(BC8571519B60222D)
QMURID(0000002BDA50)
URTYPE(CICS)
USERID(MQTEST)
APPLTAG(IYRCSQ1)
ASID(0000)
APPLTYPE(CICS)
TRANSID(GP02)
TASKNO(0000096)
END CONN DETAILS
```

For CICS connections, the NID consists of the CICS applid and a unique number provided by CICS at the time the syncpoint log entries are written. This unique number is stored in records written to both the CICS system log and the IBM MQ log at syncpoint processing time. This value is referred to in CICS as the *recovery token*.

2. Scan the CICS log for entries related to a particular unit of recovery.

   Look for a PREPARE record for the task-related installation where the recovery token field (JCSRMTKN) equals the value obtained from the network ID. The network ID is supplied by IBM MQ in the DISPLAY CONN command output.

   The PREPARE record in the CICS log for the units of recovery provides the CICS task number. All other entries on the log for this CICS task can be located using this number.

   You can use the CICS journal print utility DFHJUP when scanning the log. For details of using this program, see the *CICS Operations and Utilities Guide*.
3. Scan the IBM MQ log for records with the NID related to a particular unit of recovery. Then use the URID from this record to obtain the rest of the log records for this unit of recovery.

   When scanning the IBM MQ log, note that the IBM MQ startup message CSQJ001I provides the start RBA for this session.

   The print log records program (CSQ1LOGP) can be used for that purpose.
4. If you need to, do in-doubt resolution in IBM MQ.

   IBM MQ can be directed to take the recovery action for a unit of recovery using an IBM MQ RESOLVE INDOUBT command.

   To recover all threads associated with a specific *connection-name*, use the NID(*) option.

   The command produces one of the following messages showing whether the thread is committed or backed out:

```
CSQV414I +CSQ1 THREAD network-id COMMIT SCHEDULED
CSQV415I +CSQ1 THREAD network-id ABORT SCHEDULED
```

When performing in-doubt resolution, CICS and the adapter are not aware of the commands to IBM MQ to commit or back out units of recovery, because only IBM MQ resources are affected. However, CICS keeps details about the in-doubt threads that could not be resolved by IBM MQ. This information is purged either when the list presented is empty, or when the list does not include a unit of recovery of which CICS has details.

Recovering IMS units of recovery manually:

Use this topic to understand what happens when the IMS adapter restarts, and then explains how to deal with any unresolved units of recovery that arise.

What happens when the IMS adapter restarts

Whenever the connection to IBM MQ is restarted, either following a queue manager restart or an IMS /START SUBSYS command, IMS initiates the following resynchronization process:

1. IMS presents the list of unit of work (UOW) IDs that it believes are in doubt to the IBM MQ IMS adapter one at a time with a resolution parameter of Commit or Backout.
2. The IMS adapter passes the resolution request to IBM MQ and reports the result back to IMS.
3. Having processed all the IMS resolution requests, the IMS adapter gets from IBM MQ a list of all UOWs that IBM MQ still holds in doubt that were initiated by the IMS system. These are reported to the IMS master terminal in message CSQQ008I.

Note: While a UOW is in doubt, any associated IBM MQ message is locked by IBM MQ and is not available to any application.

How to resolve IMS units of recovery manually

When IMS connects to IBM MQ, IBM MQ might have one, or more in-doubt units of recovery that have not been resolved.

If IBM MQ has in-doubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal:

```
CSQQ008I nn units of recovery are still in doubt in queue manager qmgr-name
```

If this message is issued, IMS was either cold-started or it was started with an incomplete log tape. This message can also be issued if IBM MQ or IMS terminates abnormally because of a software error or other subsystem failure.

After receiving the CSQQ008I message:
- The connection remains active.
- IMS applications can still access IBM MQ resources.
- Some IBM MQ resources remain locked out.

If the in-doubt thread is not resolved, IMS message queues can start to build up. If the IMS queues fill to capacity, IMS terminates. You must be aware of this potential difficulty, and you must monitor IMS until the in-doubt units of recovery are fully resolved.

Recovery procedure

Use the following procedure to recover the IMS units of work:

1. Force the IMS log closed, using /SWI OLDS, and then archive the IMS log. Use the utility, DFSERA10, to print the records from the previous IMS log tape. Type X'3730' log records indicate a phase-2 commit request and type X'38' log records indicate an abort request. Record the requested action for the last transaction in each dependent region.
2. Run the DL/I batch job to back out each PSB involved that has not reached a commit point. The process might take some time because transactions are still being processed. It might also lock up a number of records, which could affect the rest of the processing and the rest of the message queues.
3. Produce a list of the in-doubt units of recovery from IBM MQ using the following command:

```
+CSQ1 DISPLAY CONN(*) WHERE(UOWSTATE EQ UNRESOLVED)
```

You receive the following message:

```
CSQM201I +CSQ1 CSQMDRTC DISPLAY CONN DETAILS
CONN(BC45A794C4290001)
EXTCONN(C3E2D8C3E2C5C3F24040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2005-02-15)
UOWLOGTI(16.39.43)
UOWSTDA(2005-02-15)
UOWSTTI(16.39.43)
UOWSTATE(UNRESOLVED)
NID(IM8F  .BC45A794D3810344)
EXTURID(
0000052900000000
)
QMURID(00000354B76E)
URTYPE(IMS)
USERID(STCPI)
APPLTAG(IM8F)
ASID(0000)
APPLTYPE(IMS)
PSTID(0004)
PSBNAME(GP01MPP)
```

For IMS, the NID consists of the IMS connection name and a unique number provided by IMS. The value is referred to in IMS as the *recovery token*. For more information, see the *IMS Customization Guide*.

4. Compare the NIDs (IMSID plus OASN in hexadecimal) displayed in the DISPLAY THREAD messages with the OASNs (4 bytes decimal) shown in the DFSERA10 output. Decide whether to commit or back out.
5. Perform in-doubt resolution in IBM MQ with the RESOLVE INDOUBT command, as follows:

```
RESOLVE INDOUBT( connection-name )
ACTION(COMMIT|BACKOUT)
NID( network-id )
```

To recover all threads associated with *connection-name*, use the NID(*) option. The command results in one of the following messages to indicate whether the thread is committed or backed out:

```
CSQV414I  THREAD network-id COMMIT SCHEDULED
CSQV415I  THREAD network-id BACKOUT SCHEDULED
```

When performing in-doubt resolution, IMS and the adapter are not aware of the commands to IBM MQ to commit or back out in-doubt units of recovery because only IBM MQ resources are affected.

Recovering RRS units of recovery manually:

Use this topic to understand the how to determine if there are in-doubt RRS units of recovery, and how to manually resolve those units of recovery.

When RRS connects to IBM MQ, IBM MQ might have one, or more in-doubt units of recovery that have not been resolved. If IBM MQ has in-doubt units of recovery that RRS did not resolve, one of the following messages is issued at the z/OS console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

Both IBM MQ and RRS provide tools to display information about in-doubt units of recovery, and techniques for manually resolving them.

In IBM MQ, use the DISPLAY CONN command to display information about in-doubt IBM MQ threads. The output from the command includes RRS unit of recovery IDs for those IBM MQ threads that have RRS as a coordinator. This can be used to determine the outcome of the unit of recovery.

Use the RESOLVE INDOUBT command to resolve the IBM MQ in-doubt thread manually. This command can be used to either commit or back out the unit of recovery after you have determined what the correct decision is.

Recovering units of recovery on another queue manager in the queue-sharing group:

Use this topic to identify, and manually recover units of recovery on other queue managers in a queue-sharing group.

If a queue manager that is a member of a queue-sharing group fails and cannot be restarted, other queue managers in the group can perform peer recovery, and take over from it. However, the queue manager might have in-doubt units of recovery that cannot be resolved by peer recovery because the final disposition of that unit of recovery is known only to the failed queue manager. These units of recovery are resolved when the queue manager is eventually restarted, but until then, they remain in doubt.

This means that certain resources (for example, messages) might be locked, making them unavailable to other queue managers in the group. In this situation, you can use the DISPLAY THREAD command to display these units of work on the inactive queue manager. If you want to resolve these units of recovery manually to make the messages available to other queue managers in the group, you can use the RESOLVE INDOUBT command.

When you issue the DISPLAY THREAD command to display units of recovery that are in doubt, you can use the QMNAME keyword to specify the name of the inactive queue manager. For example, if you issue the following command:

```
+CSQ1 DISPLAY THREAD(*) TYPE(INDOUBT) QMNAME(QM01)
```

You receive the following messages:

```
CSQV436I +CSQ1 INDOUBT THREADS FOR QM01 -
NAME   THREAD-XREF    URID NID
USER1  000000000000000000000000 CSQ:0001.0
USER2  000000000000000000000000 CSQ:0002.0
DISPLAY THREAD REPORT COMPLETE
```

If the queue manager specified is active, IBM MQ does not return information about in-doubt threads, but issues the following message:

```
CSQV435I CANNOT USE QMNAME KEYWORD, QM01 IS ACTIVE
```

Use the IBM MQ command RESOLVE INDOUBT to resolve the in-doubt threads manually. Use the QMNAME keyword to specify the name of the inactive queue manager in the command.

This command can be used to commit or back out the unit of recovery. The command resolves the shared portion of the unit of recovery only; any local messages are unaffected and remain locked until the queue manager restarts, or reconnects to CICS, IMS, or RRS batch.

IBM MQ and IMS

IBM MQ provides two components to interface with IMS, the IBM MQ - IMS adapter, and the IBM MQ - IMS bridge. These components are commonly called the IMS adapter, and the IMS bridge.

Operating the IMS adapter

Use this topic to understand how to operate the IMS adapter, which connects IBM MQ to IMS systems.

Note: The IMS adapter does not incorporate any operations and control panels.

This topic contains the following sections:
- "Controlling IMS connections" on page 363
- "Connecting from the IMS control region" on page 363
- "Displaying in-doubt units of recovery" on page 365
- "Controlling IMS dependent region connections" on page 367
- "Disconnecting from IMS" on page 370
- "Controlling the IMS trigger monitor" on page 370

Controlling IMS connections:

Use this topic to understand the IMS operator commands which control and monitor the connection to IBM MQ.

IMS provides the following operator commands to control and monitor the connection to IBM MQ:

/CHANGE SUBSYS
: Deletes an in-doubt unit of recovery from IMS.

/DISPLAY OASN SUBSYS
: Displays outstanding recovery elements.

/DISPLAY SUBSYS
: Displays connection status and thread activity.

/START SUBSYS
: Connects the IMS control region to a queue manager.

/STOP SUBSYS
: Disconnects IMS from a queue manager.

/TRACE
: Controls the IMS trace.

For more information about these commands, see the *IMS/ESA® Operator's Reference* manual for the level of IMS that you are using.

IMS command responses are sent to the terminal from which the command was issued. Authorization to issue IMS commands is based on IMS security.

Connecting from the IMS control region:

Use this topic to understand the mechanisms available to connect from IMS to IBM MQ.

IMS makes one connection from its control region to each queue manager that uses IMS. IMS must be enabled to make the connection in one of these ways:
- Automatically during either:
  - A cold start initialization.
  - A warm start of IMS, if the IBM MQ connection was active when IMS was shut down.
- In response to the IMS command:

```
/START SUBSYS sysid
```

  where `sysid` is the queue manager name.

  The command can be issued regardless of whether the queue manager is active.

The connection is not made until the first MQ API call to the queue manager is made. Until that time, the IMS command /DIS SUBSYS shows the status as 'NOT CONN'.

The order in which you start IMS and the queue manager is not significant.

IMS cannot re-enable the connection to the queue manager automatically if the queue manager is stopped with a STOP QMGR command, the IMS command /STOP SUBSYS, or an abnormal end. Therefore, you must make the connection by using the IMS command /START SUBSYS.

Initializing the adapter and connecting to the queue manager

The adapter is a set of modules loaded into the IMS control and dependent regions, using the IMS external Subsystem Attach Facility.

This procedure initializes the adapter and connects to the queue manager:

1. Read the subsystem member (SSM) from IMS.PROCLIB. The SSM chosen is an IMS EXEC parameter. There is one entry in the member for each queue manager to which IMS can connect. Each entry contains control information about an IBM MQ adapter.
2. Load the IMS adapter.

   Note: IMS loads one copy of the adapter modules for each IBM MQ instance that is defined in the SSM member.
3. Attach the external subsystem task for IBM MQ.
4. Run the adapter with the CTL EXEC parameter (IMSID) as the connection name.

The process is the same whether the connection is part of initialization or a result of the IMS command /START SUBSYS.

If the queue manager is active when IMS tries to make the connection, the following messages are sent:

- to the z/OS console:

```
DFS3613I ESS TCB INITIALIZATION COMPLETE
```

- to the IMS master terminal:

```
CSQQ000I IMS/TM imsid connected to queue manager ssnm
```

When IMS tries to make the connection and *the queue manager is not active*, the following messages are sent to the IMS master terminal each time an application makes an MQI call:

```
CSQQ001I IMS/TM imsid not connected to queue manager ssnm.
Notify message accepted
DFS3607I MQM1   SUBSYSTEM ID EXIT FAILURE, FC = 0286, RC = 08,
JOBNAME = IMSEMPR1
```

If you get DFS3607I messages when you start the connection to IMS or on system startup, this indicates that the queue manager is not available. To prevent a large number of messages being generated, you must do one of the following:

1. Start the relevant queue manager.
2. Issue the IMS command:

```
/STOP SUBSYS
```

so that IMS does not expect to connect to the queue manager.

If you do neither, a DFS3607I message and the associated CSQQ001I message are issued each time a job is scheduled in the region and each time a connection request to the queue manager is made by an application.

Thread attachment

In an MPP or IFP region, IMS makes a thread connection when the first application program is scheduled into that region, even if that application program does not make an IBM MQ call. In a BMP region, the thread connection is made when the application makes its first IBM MQ call (**MQCONN** or **MQCONNX**). This thread is retained for the duration of the region or until the connection is stopped.

For both the message driven and non-message driven regions, the recovery thread cross-reference identifier, *Thread-xref*, associated with the thread is:

```
PSTid + PSBname
```

where:

PSTid Partition specification table region identifier

PSBname
Program specification block name

You can use connection IDs as unique identifiers in IBM MQ commands, in which case IBM MQ automatically inserts these IDs into any operator message that it generates.

Displaying in-doubt units of recovery:

You can display and in-doubt of units of recovery and attempt to recover them.

The operational steps used to list and recover in-doubt units of recovery in this topic are for relatively simple cases only. If the queue manager ends abnormally while connected to IMS, IMS might commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*. A decision must be made about the status of the work.

To display a list of in-doubt units of recovery, issue the command:

```
+CSQ1  DISPLAY CONN(*) WHERE(UOWSTATE EQ UNRESOLVED)
```

IBM MQ responds with a message like the following:

For an explanation of the attributes in this message, see the description of the DISPLAY CONN

```
CSQM201I +CSQ1 CSQMDRTC DIS CONN DETAILS
CONN(BC0F6125F5A30001)
EXTCONN(C3E2D8C3C3E2D8F14040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2004-11-02)
UOWLOGTI(12.27.58)
UOWSTDA(2004-11-02)
UOWSTTI(12.27.58)
UOWSTATE(UNRESOLVED)
NID(CSQ1CHIN.BC0F5F1C86FC0766)
EXTURID(000000000000001F000000007472616E5F6964547565204E6F762020...)
QMURID(000000026232)
URTYPE(XA)
USERID( )
APPLTAG(CSQ1CHIN)
ASID(0000)
APPLTYPE(CHINIT)
CHANNEL( )
CONNAME( )
END CONN DETAILS
```

command.

Recovering in-doubt units of recovery

To recover in-doubt units of recovery, issue this command:

```
+CSQ1  RESOLVE INDOUBT( connection-name ) ACTION(COMMIT|BACKOUT)
NID( net-node.number )
```

where:

connection-name
    The IMS system ID.

ACTION
    Indicates whether to commit (COMMIT) or back out (BACKOUT) this unit of recovery.

net-node.number
    The associated net-node.number.

When you have issued the RESOLVE INDOUBT command, one of the following messages is displayed:

```
CSQV414I +CSQ1 THREAD network-id COMMIT SCHEDULED

CSQV415I +CSQ1 THREAD network-id BACKOUT SCHEDULED
```

Resolving residual recovery entries

At given times, IMS builds a list of residual recovery entries (RREs). RREs are units of recovery about which IBM MQ might be in doubt. They arise in several situations:

- If the queue manager is not active, IMS has RREs that cannot be resolved until the queue manager is active. These RREs are not a problem.
- If the queue manager is active and connected to IMS, and if IMS backs out the work that IBM MQ has committed, the IMS adapter issues message CSQQ010E. If the data in the two systems must be consistent, there is a problem. For information about resolving this problem, see "Recovering IMS units of recovery manually" on page 358.
- If the queue manager is active and connected to IMS, there might still be RREs even though no messages have informed you of this problem. After the IBM MQ connection to IMS has been established, you can issue the following IMS command to find out if there is a problem:

```
/DISPLAY OASN SUBSYS sysid
```

To purge the RRE, issue one of the following IMS commands:

```
/CHANGE SUBSYS sysid RESET
/CHANGE SUBSYS sysid RESET OASN nnnn
```

where *nnnn* is the originating application sequence number listed in response to your `+CSQ1` DISPLAY command. This is the schedule number of the program instance, giving its place in the sequence of invocations of that program since the last IMS cold start. IMS cannot have two in-doubt units of recovery with the same schedule number.

These commands reset the status of IMS ; they do not result in any communication with IBM MQ.

Controlling IMS dependent region connections:

You can control, monitor, and, when necessary, terminate connections between IMS and IBM MQ.

Controlling IMS dependent region connections involves the following activities:
- Connecting from dependent regions
- Region error options
- Monitoring the activity on connections
- Disconnecting from dependent regions

Connecting from dependent regions

The IMS adapter used in the control region is also loaded into dependent regions. A connection is made from each dependent region to IBM MQ. This connection is used to coordinate the commitment of IBM MQ and IMS work. To initialize and make the connection, IMS does the following:

1. Reads the subsystem member (SSM) from IMS.PROCLIB.

   A subsystem member can be specified on the dependent region EXEC parameter. If it is not specified, the control region SSM is used. If the region is never likely to connect to IBM MQ, to avoid loading the adapter, specify a member with no entries.
2. Loads the IBM MQ adapter.

   For a batch message program, the load is not done until the application issues its first messaging command. At that time, IMS tries to make the connection.

   For a message-processing program region or IMS fast-path region, the attempt is made when the region is initialized.

Region error options

If the queue manager is not active, or if resources are not available when the first messaging command is sent from application programs, the action taken depends on the error option specified on the SSM entry. The options are:

R The appropriate return code is sent to the application.

Q The application ends abnormally with abend code U3051. The input message is re-queued.

A The application ends abnormally with abend code U3047. The input message is discarded.

Monitoring the activity on connections

A thread is established from a dependent region when an application makes its first successful IBM MQ request. You can display information about connections and the applications currently using them by issuing the following command from IBM MQ:

```
+CSQ1 DISPLAY CONN(*) ALL
```

The command produces a message like the following:

```
CONN(BC45A794C4290001)
EXTCONN(C3E2D8C3C3E2D8F14040404040404040)
TYPE(CONN)
CONNOPTS(
MQCNO_STANDARD_BINDING
)
UOWLOGDA(2004-12-15)
UOWLOGTI(16.39.43)
UOWSTDA(2004-12-15)
UOWSTTI(16.39.43)
UOWSTATE(ACTIVE)
NID( )
EXTURID(
0000052900000000
)
QMURID(00000354B76E)
URTYPE(IMS)
USERID(STCPI)
APPLTAG(IM8F)
ASID(0049)
APPLTYPE(IMS)
PSTID(0004)
PSBNAME(GP01MPP)
```

For the control region, `thread-xref` is the special value CONTROL. For dependent regions, it is the PSTid concatenated with the PSBname. `auth-id` is either the user field from the job card, or the ID from the z/OS started procedures table.

For an explanation of the displayed list, see the description of message CSQV402I in the IBM MQ for z/OS messages, completion, and reason codes documentation.

IMS provides a display command to monitor the connection to IBM MQ. It shows which program is active on each dependent region connection, the LTERM user name, and the control region connection status. The command is:

```
/DISPLAY SUBSYS name
```

The status of the connection between IMS and IBM MQ is shown as one of:

```
CONNECTED
NOT CONNECTED
CONNECT IN PROGRESS
STOPPED
STOP IN PROGRESS
INVALID SUBSYSTEM NAME= name
SUBSYSTEM name NOT DEFINED BUT RECOVERY OUTSTANDING
```

The thread status from each dependent region is one of the following:

```
CONN
CONN, ACTIVE (includes LTERM of user)
```

Disconnecting from dependent regions

To change values in the SSM member of IMS.PROCLIB, you disconnect a dependent region. To do this, you must:

1. Issue the IMS command:

```
/STOP REGION
```

2. Update the SSM member.
3. Issue the IMS command:

```
/START REGION
```

Disconnecting from IMS:

The connection is ended when either IMS or the queue manager terminates. Alternatively, the IMS master terminal operator can explicitly break the connection.

To terminate the connection between IMS and IBM MQ, use the following IMS command:

```
/STOP SUBSYS sysid
```

The command sends the following message to the terminal that issued it, typically the master terminal operator (MTO):

```
DFS058I STOP COMMAND IN PROGRESS
```

The IMS command:

```
/START SUBSYS sysid
```

is required to reestablish the connection.

Note: The IMS command /STOP SUBSYS is not completed if an IMS trigger monitor is running.

Controlling the IMS trigger monitor:

You can use the CSQQTRMN transaction to stop, and start the IMS trigger monitor.

The IMS trigger monitor (the CSQQTRMN transaction) is described in the Setting up the IMS trigger monitor.

To control the IMS trigger monitor see:
- Starting CSQQTRMN
- Stopping CSQQTRMN

Starting CSQQTRMN
1. Start a batch-oriented BMP that runs the program CSQQTRMN for each initiation queue you want to monitor.
2. Modify your batch JCL to add a DDname of CSQQUT1 that points to a data set containing the following information:

```
QMGRNAME=q_manager_name    Comment: queue manager name
INITQUEUENAME=init_q_name   Comment: initiation queue name
LTERM=lterm          Comment: LTERM to remove error messages
CONSOLEMESSAGES=YES      Comment: Send error messages to console
```

   where:

`q_manager_name`	The name of the queue manager (if this is blank, the default nominated in CSQQDEFV is assumed)
`init_q_name`	The name of the initiation queue to be monitored
`lterm`	The IMS LTERM name for the destination of error messages (if this is blank, the default value is MASTER).
`CONSOLEMESSAGES=YES`	Requests that messages sent to the nominated IMS LTERM are also sent to the z/OS console. If this parameter is omitted or misspelled, the default is NOT to send messages to the console.

3. Add a DD name of CSQQUT2 if you want a printed report of the processing of CSQQUT1 input.

Note:
1. The data set CSQQUT1 is defined with LRECL=80. Other DCB information is taken from the data set. The DCB for data set CSQQUT2 is RECFM=VBA and LRECL=125.
2. You can put only one keyword on each record. The keyword value is delimited by the first blank following the keyword; this means that you can include comments. An asterisk in column 1 means that the whole input record is a comment.
3. If you misspell either of the `QMGRNAME` or `LTERM` keywords, CSQQTRMN uses the default for that keyword.
4. Ensure that the subsystem is started in IMS (by the /START SUBSYS command) before submitting the trigger monitor BMP job. If it is not started, your trigger monitor job terminates with abend code U3042.

Stopping CSQQTRMN

Once started, CSQQTRMN runs until either the connection between IBM MQ and IMS is broken due to one of the following events:
- the queue manager ending
- IMS ending

or a z/OS STOP **`jobname`** command is entered.

Controlling the IMS bridge

Use this topic to understand the IMS commands that you can use to control the IMS bridge.

There are no IBM MQ commands to control the IBM MQ-IMS bridge. However, you can stop messages being delivered to IMS in the following ways:
- For non-shared queues, by using the ALTER QLOCAL(xxx) GET(DISABLED) command for all bridge queues.
- For clustered queues, by using the SUSPEND QMGR CLUSTER(xxx) command. This is effective only when another queue manager is also hosting the clustered bridge queue.
- For clustered queues, by using the SUSPEND QMGR FACILITY(IMSBRIDGE) command. No further messages are sent to IMS, but the responses for any outstanding transactions are received from IMS.

  To start sending messages to IMS again, issue the RESUME QMGR FACILITY(IMSBRIDGE) command.

You can also use the MQSC command DISPLAY SYSTEM to display whether the bridge is suspended.

See MQSC commands for details of these commands.

For further information see:
- “Starting and stopping the IMS bridge” on page 372
- “Controlling IMS connections” on page 372

- Controlling bridge queues
- "Resynchronizing the IMS bridge" on page 373
- Working with tpipe names
- Deleting messages from IMS
- Deleting tpipes
- "IMS Transaction Expiration" on page 375

Starting and stopping the IMS bridge

Start the IBM MQ bridge by starting OTMA. Either use the IMS command:

```
/START OTMA
```

or start it automatically by specifying OTMA=YES in the IMS system parameters. If OTMA is already started, the bridge starts automatically when queue manager startup has completed. An IBM MQ event message is produced when OTMA is started.

Use the IMS command:

```
/STOP OTMA
```

to stop OTMA communication. When this command is issued, an IBM MQ event message is produced.

Controlling IMS connections

IMS provides these operator commands to control and monitor the connection to IBM MQ:

/DEQUEUE TMEMBER *tmember* **TPIPE** *tpipe*
: Removes messages from a Tpipe. Specify PURGE to remove all messages or PURGE1 to remove the first message only.

/DISPLAY OTMA
: Displays summary information about the OTMA server and clients, and client status.

/DISPLAY TMEMBER *name*
: Displays information about an OTMA client.

/DISPLAY TRACE TMEMBER *name*
: Displays information about what is being traced.

/SECURE OTMA
: Sets security options.

/START OTMA
: Enables communications through OTMA.

/START TMEMBER *tmember* **TPIPE** *tpipe*
: Starts the named Tpipe.

/STOP OTMA
: Stops communications through OTMA.

/STOP TMEMBER *tmember* **TPIPE** *tpipe*
: Stops the named Tpipe.

/TRACE
: Controls the IMS trace.

For more information about these commands, see the *IMS/ESA Operators Reference* manual for the level of IMS that you are using.

IMS command responses are sent to the terminal from which the command was issued. Authorization to issue IMS commands is based on IMS security.

Controlling bridge queues

To stop communicating with the queue manager with XCF member name *tmember* through the bridge, issue the following IMS command:

```
/STOP TMEMBER tmember TPIPE ALL
```

To resume communication, issue the following IMS command:

```
/START TMEMBER tmember TPIPE ALL
```

The Tpipes for a queue can be displayed using the MQ DISPLAY QUEUE command.

To stop communication with the queue manager on a single Tpipe, issue the following IMS command:

```
/STOP TMEMBER tmember TPIPE tpipe
```

One or two Tpipes are created for each active bridge queue, so issuing this command stops communication with the IBM MQ queue. To resume communication, use the following IMS command :

```
/START TMEMBER tmember TPIPE tpipe
```

Alternatively, you can alter the attributes of the IBM MQ queue to make it get inhibited.

Resynchronizing the IMS bridge

The IMS bridge is automatically restarted whenever the queue manager, IMS, or OTMA are restarted.

The first task undertaken by the IMS bridge is to resynchronize with IMS. This involves IBM MQ and IMS checking sequence numbers on every synchronized Tpipe. A synchronized Tpipe is used when persistent messages are sent to IMS from an IBM MQ - IMS bridge queue using commit mode zero (commit-then-send).

If the bridge cannot resynchronize with IMS, the IMS sense code is returned in message CSQ2023E and the connection to OTMA is stopped. If the bridge cannot resynchronize with an individual IMS Tpipe, the IMS sense code is returned in message CSQ2025E and the Tpipe is stopped. If a Tpipe has been cold started, the recoverable sequence numbers are automatically reset to 1.

If the bridge discovers mismatched sequence numbers when resynchronizing with a Tpipe, message CSQ2020E is issued. Use the IBM MQ command RESET TPIPE to initiate resynchronization with the IMS Tpipe. You need to provide the XCF group and member name, and the name of the Tpipe; this information is provided by the message.

You can also specify:

- A new recoverable sequence number to be set in the Tpipe for messages sent by IBM MQ, and to be set as the partner's receive sequence number. If you do not specify this, the partner's receive sequence number is set to the current IBM MQ send sequence number.
- A new recoverable sequence number to be set in the Tpipe for messages received by IBM MQ, and to be set as the partner's send sequence number. If you do not specify this, the partner's send sequence number is set to the current IBM MQ receive sequence number.

If there is an unresolved unit of recovery associated with the Tpipe, this is also notified in the message. Use the IBM MQ command RESET TPIPE to specify whether to commit the unit of recovery, or back it out. If you commit the unit of recovery, the batch of messages has already been sent to IMS, and is deleted from the bridge queue. If you back the unit of recovery out, the messages are returned to the bridge queue, to be later sent to IMS.

Commit mode 1 (send-then-commit) Tpipes are not synchronized.

Considerations for Commit mode 1 transactions

In IMS, commit mode 1 (CM1) transactions send their output replies before sync point.

A CM1 transaction might not be able to send its reply, for example because:
- The Tpipe on which the reply is to be sent is stopped
- OTMA is stopped
- The OTMA client (that is, the queue manager) has gone away
- The reply-to queue and dead-letter queue are unavailable

For these reasons, the IMS application sending the message pseudo-abends with code U0119. The IMS transaction and program are not stopped in this case.

These reasons often prevent messages being sent into IMS, as well as replies being delivered from IMS. A U0119 abend can occur if:
- The Tpipe, OTMA, or the queue manager is stopped while the message is in IMS
- IMS replies on a different Tpipe to the incoming message, and that Tpipe is stopped
- IMS replies to a different OTMA client, and that client is unavailable.

Whenever a U0119 abend occurs, both the incoming message to IMS and the reply messages to IBM MQ are lost. If the output of a CM0 transaction cannot be delivered for any of these reasons, it is queued on the Tpipe within IMS.

Working with tpipe names

Many of the commands used to control the IBM MQ - IMS bridge require the *tpipe* name. Use this topic to understand how you can find further details of the tpipe name.

You need *tpipe* names for many of the commands that control the IBM MQ - IMS bridge. You can get the tpipe names from DISPLAY QUEUE command and note the following points:
- tpipe names are assigned when a local queue is defined
- a local queue is given two tpipe names, one for sync and one for non-sync
- tpipe names will not be known to IMS until after some communication between IMS and IBM MQ specific to that particular local queue takes place
- For a tpipe to be available for use by the IBM MQ - IMS bridge its associated queue must be assigned to a Storage Class that has the correct XCF group and member name fields completed

Deleting messages from IMS

A message that is destined for IBM MQ through the IMS bridge can be deleted if the Tmember/Tpipe is stopped. To delete one message for the queue manager with XCF member name *tmember*, issue the following IMS command:

```
/DEQUEUE TMEMBER tmember TPIPE tpipe PURGE1
```

To delete all the messages on the Tpipe, issue the following IMS command:

```
/DEQUEUE TMEMBER tmember TPIPE tpipe PURGE
```

Deleting tpipes

You cannot delete IMS tpipes yourself. They are deleted by IMS at the following times:
- Synchronized tpipes are deleted when IMS is cold started.
- Non-synchronized tpipes are deleted when IMS is restarted.

IMS Transaction Expiration

An expiration time is associated with a transaction; any IBM MQ message can have an expiration time associated with it. The expiration interval is passed from the application, to IBM MQ, using the MQMD.Expiry field. The time is the duration of a message before it expires, expressed as a value in tenths of a second. An attempt to perform the MQGET of a message, later than it has expired, results in the message being removed from the queue and expiry processing performed. The expiration time decreases as a message flows between queue managers on an IBM MQ network. When an IMS message is passed across the IMS bridge to OTMA, the remaining message expiry time is passed to OTMA as a transaction expiration time.

If a transaction has an expiration time specified, OTMA expires the input transactions in three different places in IMS:
- input message receiving from XCF
- input message enqueuing time
- application GU time

No expiration is performed after the GU time.

The transaction EXPRTIME can be provided by:
- IMS transaction definition
- IMS OTMA message header
- IMS DFSINSX0 user exit
- IMS `CREATE` or `UPDATE TRAN` commands

IMS indicates that it has expired a transaction by abending a transaction with 0243, and issuing a message. The message issued is either DFS555I in the non-shared-queues environment, or DFS2224I in the shared-queues environment.

Operating IBM MQ Advanced Message Security

To enter commands for the IBM MQ Advanced Message Security address space, use the z/OS MODIFY command.

For example,

```
F qmgr AMSM, cmd
```

The following MODIFY commands are accepted:

Table 15. IBM MQ Advanced Message Security address space MODIFY commands

Command	Option	Description
DISPLAY		Display version information
REFRESH	KEYRING POLICY ALL	Refresh the key ring certificates, security policies, or both.
SMFAUDIT	SUCCESS FAILURE ALL	Set whether SMF auditing is required when AMS successfully protects/unprotects messages, when AMS fails to protect/unprotect messages, or both.
SMFTYPE	0 - 255	Set the SMF record type to be generated when AMS protects/unprotects messages. To disable SMF auditing specify a record type of 0.

Note: To specify an option it must be separated by a comma. For example:

```
F qmgrAMSM,REFRESH KEYRING
F qmgrAMSM,SMFAUDIT ALL
F qmgrAMSM,SMFTYPE 180
```

REFRESH command.

An application issuing an MQOPEN call will pick up the changes. Existing applications continue to use the options from when that application opened the queue. To pick up the changes, an application has to close and reopen the queue.

Security

Security is an important consideration for both developers of IBM MQ applications, and for system administrators configuring IBM MQ authorities.

Security overview

This collection of topics introduces the IBM MQ security concepts.

Security concepts and mechanisms, as they apply to any computer system, are presented first, followed by a discussion of those security mechanisms as they are implemented in IBM MQ.

Security concepts and mechanisms

This collection of topics describes aspects of security to consider in your IBM MQ installation.

The commonly accepted aspects of security are as follows:
- “Identification and authentication” on page 378
- “Authorization” on page 379
- “Auditing” on page 379
- “Confidentiality” on page 379
- “Data integrity” on page 379

Security mechanisms are technical tools and techniques that are used to implement security services. A mechanism might operate by itself, or with others, to provide a particular service. Examples of common security mechanisms are as follows:
- “Cryptography” on page 380
- “Message digests and digital signatures” on page 382
- “Digital certificates” on page 382
- “Public Key Infrastructure (PKI)” on page 387

When you are planning an IBM MQ implementation, consider which security mechanisms you require to implement those aspects of security that are important to you. For information about what to consider after you have read these topics, see “Planning for your security requirements” on page 443.

Related concepts:

"Working with SSL or TLS" on page 658
These topics give instructions for performing single tasks related to using SSL or TLS with IBM MQ.

Related information:

Connecting two queue managers using SSL or TLS

Identification and authentication

Identification is the ability to identify uniquely a user of a system or an application that is running in the system. *Authentication* is the ability to prove that a user or application is genuinely who that person or what that application claims to be.

For example, consider a user who logs on to a system by entering a user ID and password. The system uses the user ID to identify the user. The system authenticates the user at the time of logon by checking that the supplied password is correct.

Non-repudiation

The *non-repudiation* service can be viewed as an extension to the identification and authentication service. In general, non-repudiation applies when data is transmitted electronically; for example, an order to a stock broker to buy or sell stock, or an order to a bank to transfer funds from one account to another.

The overall goal of the non-repudiation service is to be able to prove that a particular message is associated with a particular individual.

The non-repudiation service can contain more than one component, where each component provides a different function. If the sender of a message ever denies sending it, the non-repudiation service with *proof of origin* can provide the receiver with undeniable evidence that the message was sent by that particular individual. If the receiver of a message ever denies receiving it, the non-repudiation service with *proof of delivery* can provide the sender with undeniable evidence that the message was received by that particular individual.

In practice, proof with virtually 100% certainty, or undeniable evidence, is a difficult goal. In the real world, nothing is fully secure. Managing security is more concerned with managing risk to a level that is acceptable to the business. In such an environment, a more realistic expectation of the non-repudiation service is to be able to provide evidence that is admissible, and supports your case, in a court of law.

Non-repudiation is a relevant security service in an IBM MQ environment because IBM MQ is a means of transmitting data electronically. For example, you might require contemporaneous evidence that a particular message was sent or received by an application associated with a particular individual.

IBM MQ with IBM MQ Advanced Message Security does not provide a non-repudiation service as part of its base function. However, this product documentation does contain suggestions on how you might provide your own non-repudiation service within an IBM MQ environment by writing your own exit programs.

Related concepts:

"Identification and authentication in IBM MQ" on page 395
In IBM MQ, you can implement identification and authentication using message context information and mutual authentication.

Authorization

Authorization protects critical resources in a system by limiting access only to authorized users and their applications. It prevents the unauthorized use of a resource or the use of a resource in an unauthorized manner.

Related concepts:

"Authorization in IBM MQ" on page 395
You can use authorization to limit what particular individuals or applications can do in your IBM MQ environment.

Auditing

Auditing is the process of recording and checking events to detect whether any unexpected or unauthorized activity has taken place, or whether any attempt has been made to perform such activity.

For more information on how you set up authorization, see "Planning authorization" on page 447 and the associated sub-topics.

Related concepts:

"Auditing in IBM MQ" on page 396
IBM MQ can issue event messages to record that unusual activity has taken place.

Confidentiality

The *confidentiality* service protects sensitive information from unauthorized disclosure.

When sensitive data is stored locally, access control mechanisms might be sufficient to protect it on the assumption that the data cannot be read if it cannot be accessed. If a greater level of security is required, the data can be encrypted.

Encrypt sensitive data when it is transmitted over a communications network, especially over an insecure network such as the Internet. In a networking environment, access control mechanisms are not effective against attempts to intercept the data, such as wiretapping.

Data integrity

The *data integrity* service detects whether there has been unauthorized modification of data.

There are two ways in which data might be altered: accidentally, through hardware and transmission errors, or because of a deliberate attack. Many hardware products and transmission protocols have mechanisms to detect and correct hardware and transmission errors. The purpose of the data integrity service is to detect a deliberate attack.

The data integrity service aims only to detect whether data has been modified. It does not aim to restore data to its original state if it has been modified.

Access control mechanisms can contribute to data integrity insofar as data cannot be modified if access is denied. But, as with confidentiality, access control mechanisms are not effective in a networking environment.

Cryptographic concepts

This collection of topics describes the concepts of cryptography applicable to IBM MQ.

The term *entity* is used to refer to a queue manager, an IBM MQ MQI client, an individual user, or any other system capable of exchanging messages.

Related concepts:

"Cryptography in IBM MQ" on page 397
IBM MQ provides cryptography by using the Secure sockets Layer (SSL) and Transport Security Layer (TLS) protocols.

Cryptography:

Cryptography is the process of converting between readable text, called *plaintext*, and an unreadable form, called *ciphertext*.

This occurs as follows:

1. The sender converts the plaintext message to ciphertext. This part of the process is called *encryption* (sometimes *encipherment*).
2. The ciphertext is transmitted to the receiver.
3. The receiver converts the ciphertext message back to its plaintext form. This part of the process is called *decryption* (sometimes *decipherment*).

The conversion involves a sequence of mathematical operations that change the appearance of the message during transmission but do not affect the content. Cryptographic techniques can ensure confidentiality and protect messages against unauthorized viewing (eavesdropping), because an encrypted message is not understandable. Digital signatures, which provide an assurance of message integrity, use encryption techniques. See "Digital signatures in SSL and TLS" on page 393 for more information.

Cryptographic techniques involve a general algorithm, made specific by the use of keys. There are two classes of algorithm:

- Those that require both parties to use the same secret key. Algorithms that use a shared key are known as *symmetric* algorithms. Figure 54 on page 381 illustrates symmetric key cryptography.
- Those that use one key for encryption and a different key for decryption. One of these must be kept secret but the other can be public. Algorithms that use public and private key pairs are known as *asymmetric* algorithms. Figure 55 on page 381 illustrates asymmetric key cryptography, which is also known as *public key cryptography*.

The encryption and decryption algorithms used can be public but the shared secret key and the private key must be kept secret.

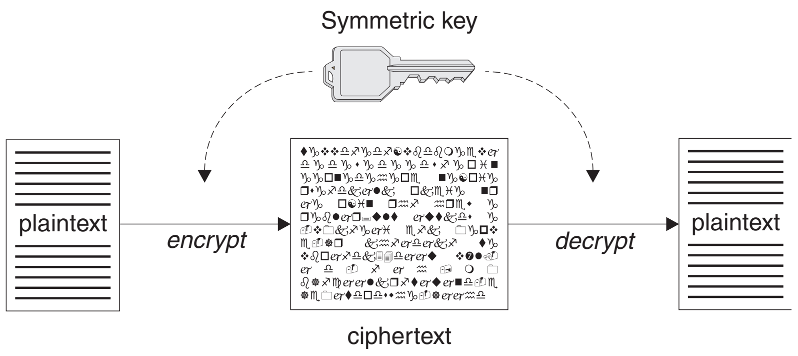


Figure 54. Symmetric key cryptography

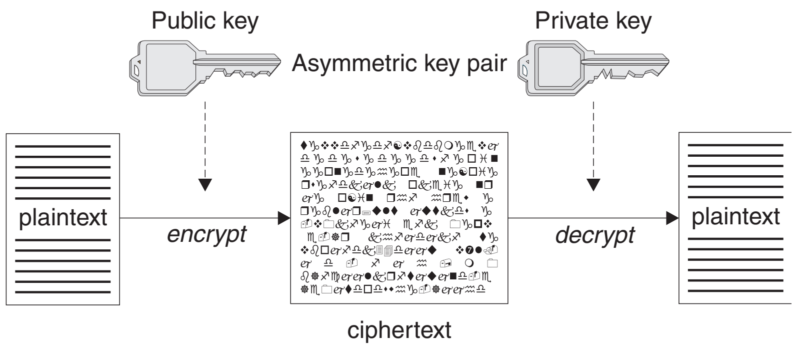


Figure 55. Asymmetric key cryptography

Figure 55 shows plaintext encrypted with the receiver's public key and decrypted with the receiver's private key. Only the intended receiver holds the private key for decrypting the ciphertext. Note that the sender can also encrypt messages with a private key, which allows anyone that holds the sender's public key to decrypt the message, with the assurance that the message must have come from the sender.

With asymmetric algorithms, messages are encrypted with either the public or the private key but can be decrypted only with the other key. Only the private key is secret, the public key can be known by anyone. With symmetric algorithms, the shared key must be known only to the two parties. This is called the *key distribution problem*. Asymmetric algorithms are slower but have the advantage that there is no key distribution problem.

Other terminology associated with cryptography is:

Strength
The strength of encryption is determined by the key size. Asymmetric algorithms require large keys, for example:

1024 bits	Low-strength asymmetric key
2048 bits	Medium-strength asymmetric key
4096 bits	High-strength asymmetric key

Symmetric keys are smaller: 256 bit keys give you strong encryption.

Block cipher algorithm
These algorithms encrypt data by blocks. For example, the RC2 algorithm from RSA Data Security Inc. uses blocks 8 bytes long. Block algorithms are typically slower than stream algorithms.

Stream cipher algorithm
These algorithms operate on each byte of data. Stream algorithms are typically faster than block algorithms.

Message digests and digital signatures:

A message digest is a fixed size numeric representation of the contents of a message. The message digest is computed by a hash function and can be encrypted, forming a digital signature.

The hash function used to compute a message digest must meet two criteria:
- It must be one way. It must not be possible to reverse the function to find the message corresponding to a particular message digest, other than by testing all possible messages.
- It must be computationally infeasible to find two messages that hash to the same digest.

The message digest is sent with the message itself. The receiver can generate a digest for the message and compare it with the digest of the sender. The integrity of the message is verified when the two message digests are the same. Any tampering with the message during transmission almost certainly results in a different message digest.

A message digest created using a secret symmetric key is known as a Message Authentication Code (MAC), because it can provide assurance that the message has not been modified.

The sender can also generate a message digest and then encrypt the digest using the private key of an asymmetric key pair, forming a digital signature. The signature must then be decrypted by the receiver, before comparing it with a locally generated digest.
Related concepts:
"Digital signatures in SSL and TLS" on page 393
A digital signature is formed by encrypting a representation of a message. The encryption uses the private key of the signatory and, for efficiency, usually operates on a message digest rather than the message itself.

Digital certificates:

Digital certificates protect against impersonation, certifying that a public key belongs to a specified entity. They are issued by a Certificate Authority.

Digital certificates provide protection against impersonation, because a digital certificate binds a public key to its owner, whether that owner is an individual, a queue manager, or some other entity. Digital certificates are also known as public key certificates, because they give you assurances about the ownership of a public key when you use an asymmetric key scheme. A digital certificate contains the public key for an entity and is a statement that the public key belongs to that entity:
- When the certificate is for an individual entity, the certificate is called a *personal certificate* or *user certificate*.
- When the certificate is for a Certificate Authority, the certificate is called a *CA certificate* or *signer certificate*.

If public keys are sent directly by their owner to another entity, there is a risk that the message could be intercepted and the public key substituted by another. This is known as a *man in the middle attack*. The solution to this problem is to exchange public keys through a trusted third party, giving you a strong assurance that the public key really belongs to the entity with which you are communicating. Instead of sending your public key directly, you ask the trusted third party to incorporate it into a digital certificate. The trusted third party that issues digital certificates is called a Certificate Authority (CA), as described in "Certificate Authorities" on page 384.

What is in a digital certificate:

Digital certificates contain specific pieces of information, as determined by the X.509 standard.

Digital certificates used by IBM MQ comply with the X.509 standard, which specifies the information that is required and the format for sending it. X.509 is the Authentication framework part of the X.500 series of standards.

Digital certificates contain at least the following information about the entity being certified:
- The owner's public key
- The owner's Distinguished Name
- The Distinguished Name of the CA that issued the certificate
- The date from which the certificate is valid
- The expiry date of the certificate
- The version number of the certificate data format as defined in X.509. The current version of the X.509 standard is Version 3, and most certificates conform to that version.
- A serial number. This is a unique identifier assigned by the CA which issued the certificate. The serial number is unique within the CA which issued the certificate: no two certificates signed by the same CA certificate have the same serial number.

An X.509 Version 2 certificate also contains an Issuer Identifier and a Subject Identifier, and an X.509 Version 3 certificate can contain a number of extensions. Some certificate extensions, such as the Basic Constraint extension, are *standard*, but others are implementation-specific. An extension can be *critical*, in which case a system must be able to recognize the field; if it does not recognize the field, it must reject the certificate. If an extension is not critical, the system can ignore it if it does not recognize it.

The digital signature in a personal certificate is generated using the private key of the CA which signed that certificate. Anyone who needs to verify the personal certificate can use the CA's public key to do so. The CA's certificate contains its public key.

Digital certificates do not contain your private key. You must keep your private key secret.

Requirements for personal certificates:

IBM MQ supports digital certificates that comply with the X.509 standard. It requires the client authentication option.

Because IBM MQ is a peer to peer system, it is viewed as client authentication in SSL terminology. Therefore, any personal certificate used for SSL authentication needs to allow a key usage of client authentication. Not all server certificates have this option enabled, so the certificate provider might need to enable client authentication on the root CA for the secure certificate.

In addition to the standards which specify the data format for a digital certificate, there are also standards for determining whether a certificate is valid. These standards have been updated over time in order to prevent certain types of security breach. For example, older X.509 version 1 and 2 certificates did not indicate whether the certificate could be legitimately used to sign other certificates. It was therefore possible for a malicious user to obtain a personal certificate from a legitimate source and create new certificates designed to impersonate other users.

When using X.509 version 3 certificates, the BasicConstraints and KeyUsage certificate extensions are used to specify which certificates can legitimately sign other certificates. The IETF RFC 5280 standard specifies a series of certificate validation rules which compliant application software must implement in order to prevent impersonation attacks. A set of certificate rules is known as a certificate validation policy.

For more information about certificate validation policies in IBM MQ, see “Certificate validation policies in IBM MQ” on page 418.

Certificate Authorities:

A Certificate Authority (CA) is a trusted third party that issues digital certificates to provide you with an assurance that the public key of an entity truly belongs to that entity.

The roles of a CA are:
- On receiving a request for a digital certificate, to verify the identity of the requestor before building, signing and returning the personal certificate
- To provide the CA's own public key in its CA certificate
- To publish lists of certificates that are no longer trusted in a Certificate Revocation List (CRL). For more information, see “Working with revoked certificates” on page 721
- To provide access to certificate revocation status by operating an OCSP responder server

Distinguished Names:

The Distinguished Name (DN) uniquely identifies an entity in an X.509 certificate.

Attention: Only the attributes in the following table can be used in an SSLPEER filter.

The following attribute types are commonly found in the DN:

SERIALNUMBER	Certificate serial number
MAIL	Email address
E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zip code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

The X.509 standard defines other attributes that do not typically form part of the DN but can provide optional extensions to the digital certificate.

The X.509 standard provides for a DN to be specified in a string format. For example:

```
CN=John Smith, OU=Test, O=IBM, C=GB
```

The Common Name (CN) can describe an individual user or any other entity, for example a web server.

The DN can contain multiple OU and DC attributes. Only one instance of each of the other attributes is permitted. The order of the OU entries is significant: the order specifies a hierarchy of Organizational Unit names, with the highest-level unit first. The order of the DC entries is also significant.

IBM MQ tolerates certain malformed DNs. For more information, see IBM MQ rules for SSLPEER values.

Related concepts:
“What is in a digital certificate” on page 383
Digital certificates contain specific pieces of information, as determined by the X.509 standard.

Obtaining personal certificates from a certificate authority:

You can obtain a certificate from a trusted external certificate authority (CA).

You obtain a digital certificate by sending information to a CA, in the form of a certificate request. The X.509 standard defines a format for this information, but some CAs have their own format. Certificate requests are typically generated by the certificate management tool your system uses; for example:

- distributed IBM i Te iKeyman tool on UNIX, Linux, and Windows systems
- z/OS RACF on z/OS.

The information contains your Distinguished Name and your public key. When your certificate management tool generates your certificate request, it also generates your private key, which you must keep secure. Never distribute your private key.

When the CA receives your request, the authority verifies your identity before building the certificate and returning it to you as a personal certificate.

Figure 56 illustrates the process of obtaining a digital certificate from a CA.

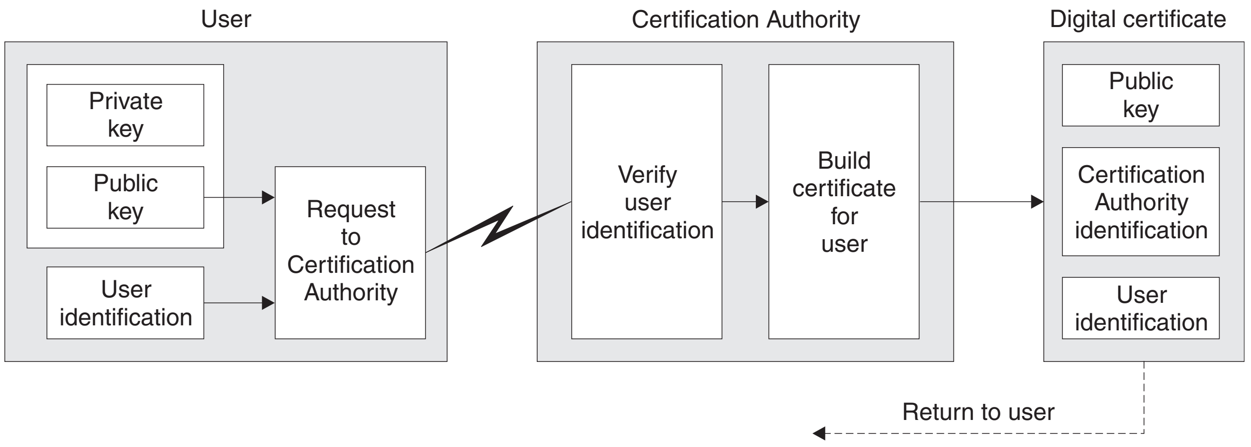


Figure 56. Obtaining a digital certificate

In the diagram:
- "User identification" includes your Subject Distinguished Name.
- "Certification Authority identification" includes the Distinguished Name of the CA that is issuing the certificate.

Digital certificates contain additional fields other than those shown in the diagram. For more information about the other fields in a digital certificate, see “What is in a digital certificate” on page 383.

How certificate chains work:

When you receive the certificate for another entity, you might need to use a *certificate chain* to obtain the *root CA* certificate.

The certificate chain, also known as the *certification path*, is a list of certificates used to authenticate an entity. The chain, or path, begins with the certificate of that entity, and each certificate in the chain is signed by the entity identified by the next certificate in the chain. The chain terminates with a root CA certificate. The root CA certificate is always signed by the certificate authority (CA) itself. The signatures of all certificates in the chain must be verified until the root CA certificate is reached.

Figure 57 illustrates a certification path from the certificate owner to the root CA, where the chain of trust begins.

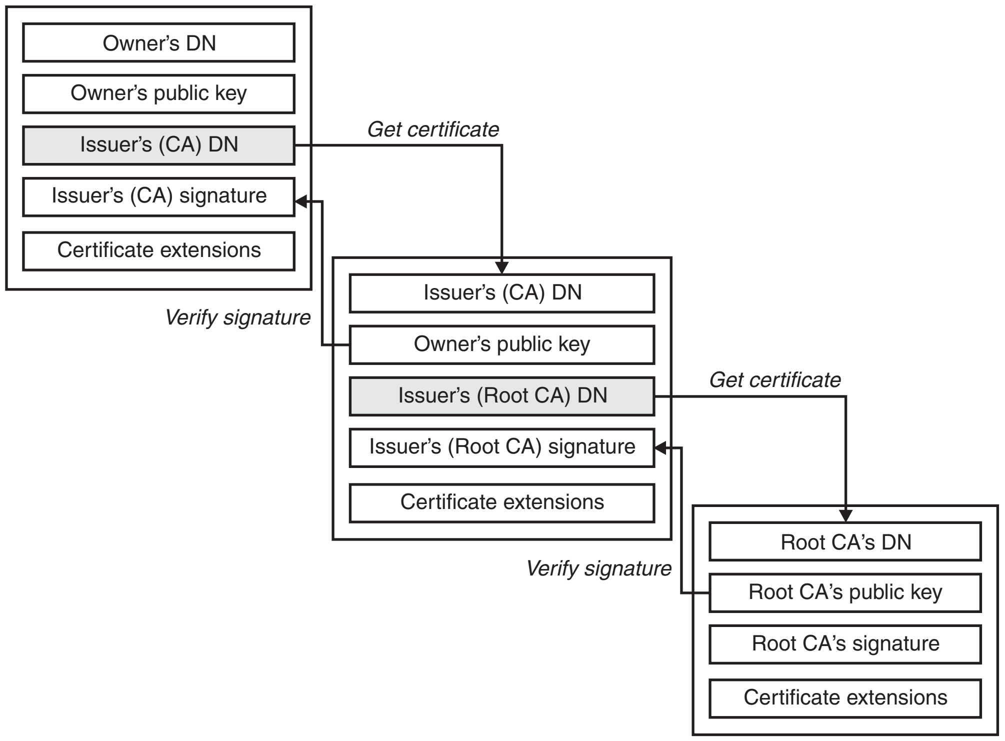


Figure 57. Chain of trust

Each certificate can contain one or more extensions. A certificate belonging to a CA typically contains a BasicConstraints extension with the isCA flag set to indicate that it is allowed to sign other certificates.

When certificates are no longer valid:

Digital certificates can expire or be revoked.

Digital certificates are issued for a fixed period and are not valid after their expiry date.

Certificates can be revoked for various reasons, including:
- The owner has moved to a different organization.
- The private key is no longer secret.

IBM MQ can check whether a certificate is revoked by sending a request to an Online Certificate Status Protocol (OCSP) responder (on UNIX, Linux and Windows systems only). Alternatively, they can access a Certificate Revocation List (CRL) on an LDAP server. The OCSP revocation and CRL information is published by a Certificate Authority. For more information, see "Working with revoked certificates" on page 721.

Public Key Infrastructure (PKI):

A Public Key Infrastructure (PKI) is a system of facilities, policies, and services that supports the use of public key cryptography for authenticating the parties involved in a transaction.

There is no single standard that defines the components of a Public Key Infrastructure, but a PKI typically comprises certificate authorities (CAs) and Registration Authorities (RAs). CAs provide the following services::
- Issuing digital certificates
- Validating digital certificates
- Revoking digital certificates
- Distributing public keys

The X.509 standards provide the basis for the industry standard Public Key Infrastructure.

Refer to "Digital certificates" on page 382 for more information about digital certificates and certificate authorities (CAs). RAs verify the information provided when digital certificates are requested. If the RA verifies that information, the CA can issue a digital certificate to the requester.

A PKI might also provide tools for managing digital certificates and public keys. A PKI is sometimes described as a *trust hierarchy* for managing digital certificates, but most definitions include additional services. Some definitions include encryption and digital signature services, but these services are not essential to the operation of a PKI.

Cryptographic security protocols: SSL and TLS

Cryptographic protocols provide secure connections, enabling two parties to communicate with privacy and data integrity. The Transport Layer Security (TLS) protocol evolved from that of the Secure Sockets Layer (SSL). IBM MQ supports both SSL and TLS.

The primary goals of both protocols is to provide confidentiality, (sometimes referred to as *privacy*), data integrity, identification, and authentication using digital certificates.

Although the two protocols are similar, the differences are sufficiently significant that SSL 3.0 and the various versions of TLS do not interoperate.

Related concepts:

"SSL and TLS security protocols in IBM MQ" on page 397
IBM MQ supports both the Transport Layer Security (TLS) and the Secure Sockets Layer (SSL) protocols to provide link level security for message channels and MQI channels.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts:

The SSL and TLS protocols enable two parties to identify and authenticate each other and communicate with confidentiality and data integrity. The TLS protocol evolved from the Netscape SSL 3.0 protocol but TLS and SSL do not interoperate.

The SSL and TLS protocols provide communications security over the internet, and allow client/server applications to communicate in a way that is confidential and reliable. The protocols have two layers: a Record Protocol and a Handshake Protocol, and these are layered above a transport protocol such as TCP/IP. They both use asymmetric and symmetric cryptography techniques.

An SSL or TLS connection is initiated by an application, which becomes the SSL or TLS client. The application which receives the connection becomes the SSL or TLS server. Every new session begins with a handshake, as defined by the SSL or TLS protocols.

A full list of CipherSpecs supported by IBM MQ is provided at "Enabling CipherSpecs" on page 797.

For more information about the SSL protocol, see the information provided at http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt. For more information about the TLS protocol, see the information provided by the TLS Working Group on the website of the Internet Engineering Task Force at http://www.ietf.org

An overview of the SSL or TLS handshake:

The SSL or TLS handshake enables the SSL or TLS client and server to establish the secret keys with which they communicate.

This section provides a summary of the steps that enable the SSL or TLS client and server to communicate with each other:

- Agree on the version of the protocol to use.
- Select cryptographic algorithms.
- Authenticate each other by exchanging and validating digital certificates.
- Use asymmetric encryption techniques to generate a shared secret key, which avoids the key distribution problem. SSL or TLS then uses the shared key for the symmetric encryption of messages, which is faster than asymmetric encryption.

For more information about cryptographic algorithms and digital certificates, refer to the related information.

This section does not attempt to provide full details of the messages exchanged during the SSL handshake. In overview, the steps involved in the SSL handshake are as follows:

1. The SSL or TLS client sends a “client hello” message that lists cryptographic information such as the SSL or TLS version and, in the client's order of preference, the CipherSuites supported by the client. The message also contains a random byte string that is used in subsequent computations. The protocol allows for the “client hello” to include the data compression methods supported by the client.
2. The SSL or TLS server responds with a “server hello” message that contains the CipherSuite chosen by the server from the list provided by the client, the session ID, and another random byte string. The server also sends its digital certificate. If the server requires a digital certificate for client authentication, the server sends a “client certificate request” that includes a list of the types of certificates supported and the Distinguished Names of acceptable Certification Authorities (CAs).
3. The SSL or TLS client verifies the server's digital certificate. For more information, see “How SSL and TLS provide identification, authentication, confidentiality, and integrity” on page 390.
4. The SSL or TLS client sends the random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The random byte string itself is encrypted with the server's public key.
5. If the SSL or TLS server sent a “client certificate request”, the client sends a random byte string encrypted with the client's private key, together with the client's digital certificate, or a “no digital certificate alert”. This alert is only a warning, but with some implementations the handshake fails if client authentication is mandatory.
6. The SSL or TLS server verifies the client's certificate. For more information, see “How SSL and TLS provide identification, authentication, confidentiality, and integrity” on page 390.
7. The SSL or TLS client sends the server a “finished” message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.
8. The SSL or TLS server sends the client a “finished” message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
9. For the duration of the SSL or TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

Figure 58 on page 390 illustrates the SSL or TLS handshake.

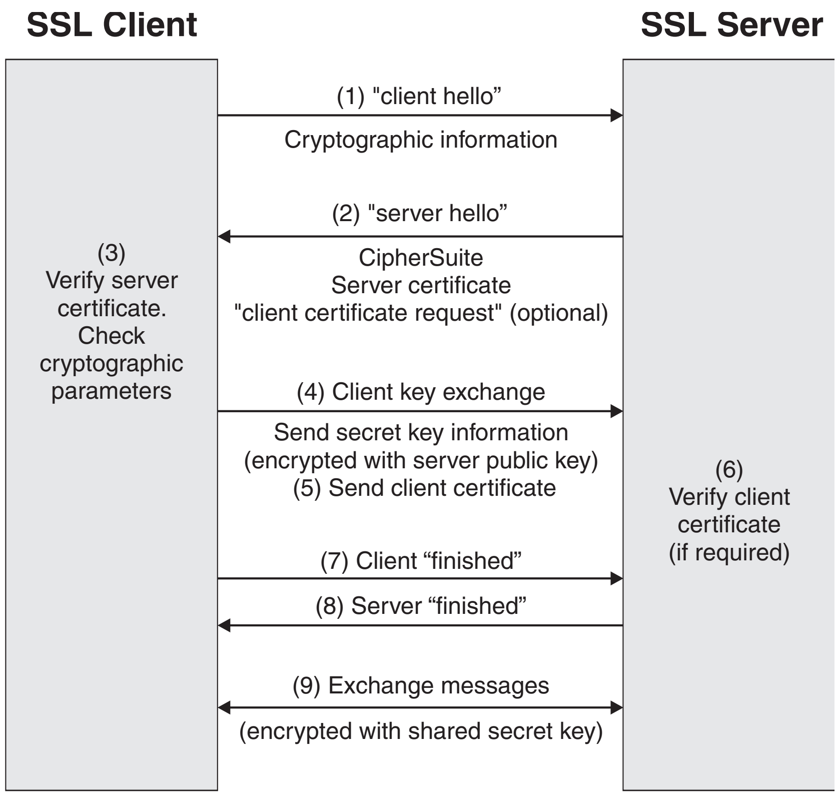

SSL Client
SSL Server
(1) "client hello”
Cryptographic information
(2) "server hello”
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client “finished”
(8) Server “finished”
(9) Exchange messages
(encrypted with shared secret key)

Figure 58. Overview of the SSL or TLS handshake

How SSL and TLS provide identification, authentication, confidentiality, and integrity:

During both client and server authentication there is a step that requires data to be encrypted with one of the keys in an asymmetric key pair and decrypted with the other key of the pair. A message digest is used to provide integrity.

How SSL and TLS provide authentication

For server authentication, the client uses the server's public key to encrypt the data that is used to compute the secret key. The server can generate the secret key only if it can decrypt that data with the correct private key.

For client authentication, the server uses the public key in the client certificate to decrypt the data the client sends during step 5 on page 389 of the handshake. The exchange of finished messages that are encrypted with the secret key (steps 7 on page 389 and 8 on page 389 in the overview) confirms that authentication is complete.

If any of the authentication steps fail, the handshake fails and the session terminates.

The exchange of digital certificates during the SSLor TLS handshake is part of the authentication process. For more information about how certificates provide protection against impersonation, refer to the related information. The certificates required are as follows, where CA `X` issues the certificate to the SSL or TLS client, and CA `Y` issues the certificate to the SSL or TLS server:

For server authentication only, the SSL or TLS server needs:
- The personal certificate issued to the server by CA `Y`
- The server's private key

and the SSL or TLS client needs:
- The CA certificate for CA Y

If the SSL or TLS server requires client authentication, the server verifies the client's identity by verifying the client's digital certificate with the public key for the CA that issued the personal certificate to the client, in this case CA X. For both server and client authentication, the server needs:
- The personal certificate issued to the server by CA Y
- The server's private key
- The CA certificate for CA X

and the client needs:
- The personal certificate issued to the client by CA X
- The client's private key
- The CA certificate for CA Y

Both the SSL or TLS server and client might need other CA certificates to form a certificate chain to the root CA certificate. For more information about certificate chains, refer to the related information.

What happens during certificate verification

As noted in steps 3 on page 389 and 6 on page 389 of the overview, the SSL or TLS client verifies the server's certificate, and the SSL or TLS server verifies the client's certificate. There are four aspects to this verification:
1. The digital signature is checked (see "Digital signatures in SSL and TLS" on page 393).
2. The certificate chain is checked; you should have intermediate CA certificates (see "How certificate chains work" on page 386).
3. The expiry and activation dates and the validity period are checked.
4. The revocation status of the certificate is checked (see "Working with revoked certificates" on page 721).

Secret key reset

During an SSL or TLS handshake a *secret key* is generated to encrypt data between the SSL or TLS client and server. The secret key is used in a mathematical formula that is applied to the data to transform plaintext into unreadable ciphertext, and ciphertext into plaintext.

The secret key is generated from the random text sent as part of the handshake and is used to encrypt plaintext into ciphertext. The secret key is also used in the MAC (Message Authentication Code) algorithm, which is used to determine whether a message has been altered. See "Message digests and digital signatures" on page 382 for more information.

If the secret key is discovered, the plaintext of a message could be deciphered from the ciphertext, or the message digest could be calculated, allowing messages to be altered without detection. Even for a complex algorithm, the plaintext can eventually be discovered by applying every possible mathematical transformation to the ciphertext. To minimize the amount of data that can be deciphered or altered if the secret key is broken, the secret key can be renegotiated periodically. When the secret key has been renegotiated, the previous secret key can no longer be used to decrypt data encrypted with the new secret key.

How SSL and TLS provide confidentiality

SSL and TLS use a combination of symmetric and asymmetric encryption to ensure message privacy. During the SSL or TLS handshake, the SSL or TLS client and server agree an encryption algorithm and a shared secret key to be used for one session only. All messages transmitted between the SSL or TLS client and server are encrypted using that algorithm and key, ensuring that the message remains private even if it is intercepted. SSL supports a wide range of cryptographic algorithms. Because SSL and TLS use asymmetric encryption when transporting the shared secret key, there is no key distribution problem. For more information about encryption techniques, refer to “Cryptography” on page 380.

How SSL and TLS provide integrity

SSL and TLS provide data integrity by calculating a message digest. For more information, refer to “Data integrity of messages” on page 809.

Use of SSL or TLS does ensure data integrity, provided that the CipherSpec in your channel definition uses a hash algorithm as described in the table in “Enabling CipherSpecs” on page 797.

In particular, if data integrity is a concern, you should avoid choosing a CipherSpec whose hash algorithm is listed as "None". Use of MD5 is also strongly discouraged as this is now very old and no longer secure for most practical purposes.

CipherSpecs and CipherSuites:

Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

A CipherSpec identifies a combination of encryption algorithm and Message Authentication Code (MAC) algorithm. Both ends of a TLS, or SSL, connection must agree on the same CipherSpec to be able to communicate.

From IBM MQ Version 8.0.0, Fix Pack 2 the SSLv3 protocol and the use of some IBM MQ CipherSpecs is deprecated. For more information, see Deprecation: SSLv3 protocol.

Important: When dealing with IBM MQ channels, you use a CipherSpec. When dealing with Java channels, JMS channels, or MQTT channels you specify a CipherSuite.

For more information about CipherSpecs, see “Enabling CipherSpecs” on page 797.

A CipherSuite is a suite of cryptographic algorithms used by a TLS or SSL connection. A suite comprises three distinct algorithms:
- The key exchange and authentication algorithm, used during the handshake
- The encryption algorithm, used to encipher the data
- The MAC (Message Authentication Code) algorithm, used to generate the message digest

There are several options for each component of the suite, but only certain combinations are valid when specified for a TLS or SSL connection. The name of a valid CipherSuite defines the combination of algorithms used. For example, the CipherSuite SSL_RSA_WITH_RC4_128_MD5 specifies:
- The RSA key exchange and authentication algorithm
- The RC4 encryption algorithm, using a 128-bit key
- The MD5 MAC algorithm

Several algorithms are available for key exchange and authentication, but the RSA algorithm is currently the most widely used. There is more variety in the encryption algorithms and MAC algorithms that are used.

Digital signatures in SSL and TLS:

A digital signature is formed by encrypting a representation of a message. The encryption uses the private key of the signatory and, for efficiency, usually operates on a message digest rather than the message itself.

Digital signatures vary with the data being signed, unlike handwritten signatures, which do not depend on the content of the document being signed. If two different messages are signed digitally by the same entity, the two signatures differ, but both signatures can be verified with the same public key, that is, the public key of the entity that signed the messages.

The steps of the digital signature process are as follows:

1. The sender computes a message digest and then encrypts the digest using the sender's private key, forming the digital signature.
2. The sender transmits the digital signature with the message.
3. The receiver decrypts the digital signature using the sender's public key, regenerating the sender's message digest.
4. The receiver computes a message digest from the message data received and verifies that the two digests are the same.

Figure 59 illustrates this process.

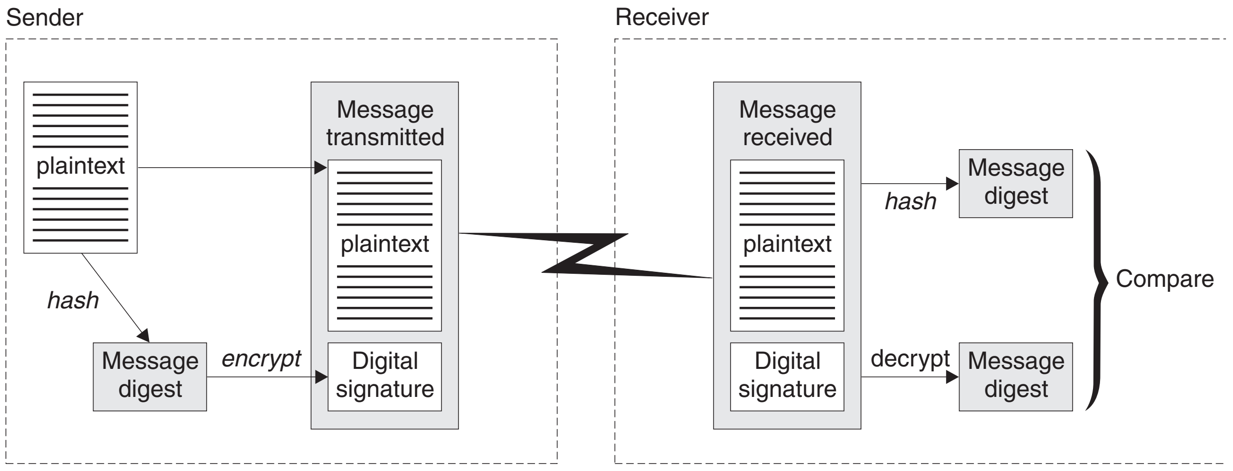


Figure 59. The digital signature process

If the digital signature is verified, the receiver knows that:

- The message has not been modified during transmission.
- The message was sent by the entity that claims to have sent it.

Digital signatures are part of integrity and authentication services. Digital signatures also provide proof of origin. Only the sender knows the private key, which provides strong evidence that the sender is the originator of the message.

Note: You can also encrypt the message itself, which protects the confidentiality of the information in the message.

Federal Information Processing Standards:

The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

A significant one of these standards is FIPS 140-2, which requires the use of strong cryptographic algorithms. FIPS 140-2 also specifies requirements for hashing algorithms to be used to protect packets against modification in transit.

IBM MQ provides FIPS 140-2 support when it has been configured to do so.

Over time, analysts develop attacks against existing encryption and hashing algorithms. New algorithms are adopted to resist those attacks. FIPS 140-2 is periodically updated to take account of these changes.

Related concepts:
"National Security Agency (NSA) Suite B Cryptography"
The government of the Unites States of America produces technical advice on IT systems and security, including data encryption. The US National Security Agency (NSA) recommends a set of interoperable cryptographic algorithms in its Suite B standard.

National Security Agency (NSA) Suite B Cryptography:

The government of the Unites States of America produces technical advice on IT systems and security, including data encryption. The US National Security Agency (NSA) recommends a set of interoperable cryptographic algorithms in its Suite B standard.

The Suite B standard specifies a mode of operation in which only a specific set of secure cryptographic algorithms are used. The Suite B standard specifies:
- The encryption algorithm (AES)
- The key exchange algorithm (Elliptic Curve Diffie-Hellman, also known as ECDH)
- The digital signature algorithm (Elliptic Curve Digital Signature Algorithm, also known as ECDSA)
- The hashing algorithms (SHA-256 or SHA-384)

Additionally, the IETF RFC 6460 standard specifies Suite B compliant profiles which define the detailed application configuration and behavior necessary to comply with the Suite B standard. It defines two profiles:
1. A Suite B compliant profile for use with TLS version 1.2. When configured for Suite B compliant operation, only the restricted set of cryptographic algorithms listed are used.
2. A transitional profile for use with TLS version 1.0 or TLS version 1.1. This profile enables interoperability with non-Suite B compliant servers. When configured for Suite B transitional operation, additional encryption and hashing algorithms may be used.

The Suite B standard is conceptually similar to FIPS 140-2, because it restricts the set of enabled cryptographic algorithms in order to provide an assured level of security.

On Windows, UNIX and Linux systems, IBM MQ, can be configured to conform to the Suite B compliant TLS 1.2 profile, but does not support the Suite B transitional profile. For further information, see "NSA Suite B Cryptography in IBM MQ" on page 415.

Related information:

"Federal Information Processing Standards" on page 394
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

IBM MQ security mechanisms

This collection of topics explains how you can implement the various security concepts in IBM MQ.

IBM MQ provides mechanisms to implement all the security concepts introduced in "Security concepts and mechanisms" on page 377. These are discussed in more detail in the following sections.

Identification and authentication in IBM MQ

In IBM MQ, you can implement identification and authentication using message context information and mutual authentication.

Here are some examples of the identification and authentication in an IBM MQ environment:

- Every message can contain *message context* information. This information is held in the message descriptor. It can be generated by the queue manager when a message is put on a queue by an application. Alternatively, the application can supply the information if the user ID associated with the application is authorized to do so.

  The context information in a message allows the receiving application to find out about the originator of the message. It contains, for example, the name of the application that put the message and the user ID associated with the application.
- When a message channel starts, it is possible for the message channel agent (MCA) at each end of the channel to authenticate its partner. This technique is known as *mutual authentication*. For the sending MCA, it provides assurance that the partner it is about to send messages to is genuine. For the receiving MCA, there is a similar assurance that it is about to receive messages from a genuine partner.

Related concepts:

"Identification and authentication" on page 378
Identification is the ability to identify uniquely a user of a system or an application that is running in the system. *Authentication* is the ability to prove that a user or application is genuinely who that person or what that application claims to be.

Authorization in IBM MQ

You can use authorization to limit what particular individuals or applications can do in your IBM MQ environment.

Here are some examples of authorization in an IBM MQ environment:

- Allowing only an authorized administrator to issue commands to manage IBM MQ resources.
- Allowing an application to connect to a queue manager only if the user ID associated with the application is authorized to do so.
- Allowing an application to open only those queues that are necessary for its function.
- Allowing an application to subscribe only to those topics that are necessary for its function.
- Allowing an application to perform only those operations on a queue that are necessary for its function. For example, an application might need only to browse messages on a particular queue, and not to put or get messages.

For more information on how you set up authorization, see "Planning authorization" on page 447 and the associated sub-topics.

Related concepts:

"Authorization" on page 379
Authorization protects critical resources in a system by limiting access only to authorized users and their applications. It prevents the unauthorized use of a resource or the use of a resource in an unauthorized manner.

Auditing in IBM MQ

IBM MQ can issue event messages to record that unusual activity has taken place.

Here are some examples of auditing in an IBM MQ environment:

- An application attempts to open a queue that it is not authorized to open. An instrumentation event message is issued. By inspecting the event message, you discover that this attempt occurred and can decide what action is necessary.
- An application attempts to open a channel, but the attempt fails because SSL does not allow the connection. An instrumentation event message is issued. By inspecting the event message, you discover that this attempt occurred and can decide what action is necessary.

Related concepts:

"Auditing" on page 379
Auditing is the process of recording and checking events to detect whether any unexpected or unauthorized activity has taken place, or whether any attempt has been made to perform such activity.

Confidentiality in IBM MQ

You can implement confidentiality in IBM MQ by encrypting messages.

Confidentiality can be ensured in an IBM MQ environment as follows:

- After a sending MCA gets a message from a transmission queue, IBM MQ uses SSL or TLS to encrypt the message before it is sent over the network to the receiving MCA. At the other end of the channel, the message is decrypted before the receiving MCA puts it on its destination queue.
- While messages are stored on a local queue, the access control mechanisms provided by IBM MQ might be considered sufficient to protect their contents against unauthorized disclosure. However, for a greater level of security, you can use IBM MQ Advanced Message Security to encrypt the messages stored in the queues.

Related concepts:

"Confidentiality" on page 379
The *confidentiality* service protects sensitive information from unauthorized disclosure.

Data integrity in IBM MQ

You can use a data integrity service to detect whether a message has been modified.

Data integrity can be ensured in an IBM MQ environment as follows:

- You can use SSL or TLS to detect whether the contents of a message have been deliberately modified while it was being transmitted over a network. In SSL and TLS, the message digest algorithm provides detection of modified messages in transit.

  All IBM MQ CipherSpecs provide a message digest algorithm, except for TLS_RSA_WITH_NULL_NULL, which does not provide message data integrity.

  IBM MQ detects modified messages upon receiving them; on receiving a modified message, IBM MQ throws an AMQ9661 error message and the channel stops.
- While messages are stored on a local queue, the access control mechanisms provided by IBM MQ might be considered sufficient to prevent deliberate modification of the contents of the messages.

  However, for a greater level of security, you can use IBM MQ Advanced Message Security to detect whether the contents of a message have been deliberately modified between the time the message was put on the queue and the time it was retrieved from the queue.

Upon detecting a modified message, the application attempting to receive the message receives a 2063 return code and, if using an MQGET call, the message is moved to the SYSTEM.PROTECTION.ERROR.QUEUE

Related concepts:

"Data integrity" on page 379
The *data integrity* service detects whether there has been unauthorized modification of data.

Cryptography in IBM MQ

IBM MQ provides cryptography by using the Secure sockets Layer (SSL) and Transport Security Layer (TLS) protocols.

For more information see "SSL and TLS security protocols in IBM MQ."

Related concepts:

"Cryptographic concepts" on page 380
This collection of topics describes the concepts of cryptography applicable to IBM MQ.

SSL and TLS security protocols in IBM MQ

IBM MQ supports both the Transport Layer Security (TLS) and the Secure Sockets Layer (SSL) protocols to provide link level security for message channels and MQI channels.

Message channels and MQI channels can use the SSL or TLS protocol to provide link level security. A caller MCA is an SSL or TLS client and a responder MCA is an SSL or TLS server. IBM MQ supports Version 3.0 of the SSL protocol and Version 1.0 and Version 1.2 of the TLS protocol. You can specify the cryptographic algorithms that are used by the SSL or protocol by supplying a CipherSpec as part of the channel definition.

Note: V 8.0.0.2 From IBM MQ Version 8.0.0, Fix Pack 2, the SSLv3 protocol and the use of some IBM MQ CipherSpecs is deprecated. For more information, see Deprecation: SSLv3 protocol.

You can use the SECPROT parameter to display the security protocol in use on a channel.

At each end of a message channel, and at the server end of an MQI channel, the MCA acts on behalf of the queue manager to which it is connected. During the SSL or TLS handshake, the MCA sends the digital certificate of the queue manager to its partner MCA at the other end of the channel. The IBM MQ code at the client end of an MQI channel acts on behalf of the user of the IBM MQ client application. During the SSL or TLS handshake, the IBM MQ code sends the user's digital certificate to the MCA at the server end of the MQI channel.

Queue managers and IBM MQ client users are not required to have personal digital certificates associated with them when they are acting as SSL or TLS clients, unless SSLCAUTH(REQUIRED) is specified at the server side of the channel.

Digital certificates are stored in a *key repository*. The queue manager attribute **`SSLKeyRepository`** specifies the location of the key repository that holds the queue manager's digital certificate. On an IBM MQ client system, the MQSSLKEYR environment variable specifies the location of the key repository that holds the user's digital certificate. Alternatively, an IBM MQ client application can specify its location in the **`KeyRepository`** field of the SSL and TLS configuration options structure, MQSCO, on an MQCONNX call. See the related topics for more information about key repositories and how to specify where they are located.

Support for SSL and TLS

IBM MQ provides support for SSL Version 3.0 and TLS 1.0 and TLS 1.2 according to the platform you are using. For more information about the SSL and TLS protocols, refer to the information in the subtopics.

IBM i
SSL and TLS support is integral to the IBM i operating system.

Java and JMS clients
These clients use the JVM to provide SSL and TLS support.

HP Integrity NonStop Server, UNIX, Linux, and Windows systems
SSL and TLS support is installed with IBM MQ.

z/OS SSL and TLS support is integral to the z/OS operating system. The SSL and TLS support on z/OS is known as *System SSL*.

For information about any prerequisites for IBM MQ SSL and TLS support, see IBM MQ System Requirements.

Related concepts:

"Cryptographic security protocols: SSL and TLS" on page 388
Cryptographic protocols provide secure connections, enabling two parties to communicate with privacy and data integrity. The Transport Layer Security (TLS) protocol evolved from that of the Secure Sockets Layer (SSL). IBM MQ supports both SSL and TLS.

The SSL or TLS key repository:

A mutually authenticated SSL or TLS connection requires a key repository at each end of the connection. The key repository includes digital certificates and private keys.

This information uses the general term *key repository* to describe the store for digital certificates and their associated private keys. The key repository is referred to by different names on different platforms and environments that support SSL and TLS:

- IBM i On IBM i: *certificate store*
- On Java and JMS: *keystore* and *truststore*
- UNIX Linux Windows On UNIX, Linux, and Windows systems: *key database file*
- z/OS On z/OS: *keyring*

For more information, see "Digital certificates" on page 382 and "Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts" on page 388.

A mutually authenticated SSL or TLS connection requires a key repository at each end of the connection. The key repository can contain the following certificates and requests:

- A number of CA certificates from various Certification Authorities that allow the queue manager or client to verify certificates that it receives from its partner at the remote end of the connection. Individual certificates might be in a certificate chain.
- One or more personal certificates received from a Certification Authority. You associate a separate personal certificate with each queue manager or IBM MQ MQI client. Personal certificates are essential on an SSL or TLS client if mutual authentication is required. If mutual authentication is not required, personal certificates are not needed on the client. The key repository might also contain the private key corresponding to each personal certificate.
- Certificate requests which are waiting to be signed by a trusted CA certificate.

For more information about protecting your key repository, see "Protecting IBM MQ key repositories" on page 399.

The location of the key repository depends on the platform you are using:

IBM i **IBM i**
The key repository is a certificate store. The default system certificate store is located at

/QIBM/UserData/ICSS/Cert/Server/Default in the integrated file system (IFS). IBM MQ stores the password for the certificate store in a *password stash file*. For example, the stash file for queue manager QM1 is /QIBM/UserData/mqm/qmgrs/QM1/ssl/Stash.sth.

Alternatively, you can specify that the IBM i system certificate store is to be used instead. To do this change the value of the queue manager **SSLKEYR** attribute to *SYSTEM. This value indicates that the queue manager must use the system certificate store, and the queue manager is registered for use as an application with Digital Certificate Manager (DCM).

The certificate store also contains the private key for the queue manager.

UNIX Linux Windows UNIX, Linux, and Windows systems

The key repository is a key database file. The name of the key database file must have a file extension of .kdb. For example, on UNIX and Linux, the default key database file for queue manager QM1 is /var/mqm/qmgrs/QM1/ssl/key.kdb. If IBM MQ is installed in the default location, the equivalent path on Windows is C:\ProgramData\IBM\MQ\Qmgrs\QM1\ssl\key.kdb.

Each key database file has an associated password stash file. This file holds encoded passwords that allow programs to access the key database. The password stash file must be in the same directory and have the same file stem as the key database, and must end with the suffix .sth, for example /var/mqm/qmgrs/QM1/ssl/key.sth

Note: PKCS #11 cryptographic hardware cards can contain the certificates and keys that are otherwise held in a key database file. When certificates and keys are held on PKCS #11 cards, IBM MQ still requires access to both a key database file and a password stash file.

On UNIX and Windows systems, the key database also contains the private key for the personal certificate associated with the queue manager or IBM MQ MQI client.

z/OS z/OS

Certificates are held in a keyring in z/OS.

Other external security managers (ESMs) also use keyrings for storing certificates.

Private keys are managed by RACF.

Protecting IBM MQ key repositories:

The key repository for IBM MQ is a file. Ensure that only the intended user can access the key repository file. This prevents an intruder or other unauthorized user copying the key repository file to another system, and then setting up an identical user ID on that system to impersonate the intended user.

The permissions on the files depend on the user's umask and which tool is used. On Windows, IBM MQ accounts require permission BypassTraverseChecking which means the permissions of the folders in the file path have no effect.

Check the file permissions of key repository files and make sure that the files and containing folder are not world readable, preferably not even group readable.

Making the keystore read-only is good practice, on whichever system you use, with only the administrator being permitted to enable write operations in order to perform maintenance.

In practice, you must protect all the keystores, whatever the location and whether they are password protected or not; protect the key repositories.

Digital certificate labels, understanding the requirements:

When setting up SSL and TLS to use digital certificates, there might be specific label requirements that you must follow, depending on the platform used and the method you use to connect.

What is the certificate label?

A certificate label is a unique identifier representing a digital certificate stored in a key repository, and provides a convenient human-readable name with which to refer to a particular certificate when performing key management functions. You assign the certificate label when adding a certificate to a key repository for the first time.

The certificate label is separate from the certificate's **`Subject Distinguished Name`** or **`Subject Common Name`** fields. Note that **`Subject Distinguished Name`** and **`Subject Common Name`** are fields within the certificate itself. These are defined when the certificate is created and cannot be changed. If necessary, however, you can change the label associated with a digital certificate.

Certificate label syntax

A certificate label can contain letters, numbers, and punctuation with the following conditions:

- distributed IBM i The certificate label can contain up to 64 characters.
- z/OS The certificate label can contain up to 32 characters.
- The certificate label cannot contain spaces.
- Labels are case sensitive.
- On systems that use EBCDIC katakana, you cannot use lowercase characters.
- IBM MQ client for HP Integrity NonStop Server certificate labels cannot contain forward-slash (/), back-slash (\), tilde (⌂), dollar ($) or more than one consecutive period (.).

Additional requirements for certificate label values are specified in the following sections.

How is the certificate label used?

IBM MQ uses certificate labels to locate a personal certificate that is sent during the SSL handshake. This eliminates ambiguity when more than one personal certificate exists in the key repository.

You can set the certificate label to a value of your choice . If you do not set a value, a default label is used that follows a naming convention depending on the platform that you are using. For details, see the sections that follow, about particular platforms.

Notes:
1. You cannot set the certificate label yourself on Java or JMS systems.
2. Auto-defined channels created by a channel automatic definition (CHAD) exit cannot set the certificate label, because the SSL or TLS handshake has occurred by the time the channel is created. Setting the certificate label in a CHAD exit for inbound channels has no effect.

In this context, an SSL or TLS client refers to the connection partner initiating the handshake, which might be an IBM MQ client or another queue manager.

During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client and the client always provides a certificate to the server if one is found. If the client is unable to locate a personal certificate, the client sends a `no certificate` response to the server.

The SSL or TLS server always validates the client certificate if one is sent. If the client does not send a certificate, authentication fails if the end of the channel that is acting as the SSL or TLS server is defined with either the **SSLCAUTH** parameter set to *REQUIRED* or an **SSLPEER** parameter value set.

Note that inbound channels (including receiver, cluster-receiver, unqualified server, and server-connection channels) only send the configured certificate if the IBM MQ version of the remote peer fully supports certificate label configuration, and the channel is using a TLS CipherSpec.

In all other cases, the queue manager **CERTLABL** parameter determines the certificate sent. In particular, the following only ever receive the certificate configured by the **CERTLABL** parameter of the queue manager, regardless of the channel-specific label setting:

- All current Java and JMS clients.
- Versions of IBM MQ prior to Version 8.0.

Additionally, the certificate used by a channel must be appropriate for the channel CipherSpec - see "Digital certificates and CipherSpec compatibility in IBM MQ" on page 419 for further information.

IBM MQ Version 8.0 supports the use of multiple certificates on the same queue manager, using a per-channel certificate label attribute. Inbound channels to the queue manager (for example, server connection or receiver) rely on detecting the channel name using TLS Server Name Indication (SNI), in order to present the correct certificate from the queue manager.

If you use MQIPT with a route that has both *SSLServer* and *SSLClient* set, there are two separate TLS sessions between the endpoints, and the SNI data will not flow across the session break.

You can use separate MQIPT routes to get multiple certificate support by selecting the appropriate certificate, for example through the *SSLServerSiteLabel* and *SSLClientSiteLabel* route properties. Alternatively, use MQIPT *SSLProxyMode* which forwards all SSL or TLS control flows intact, including the SNI name.

Note that multiple certificates across MQIPT work only if you are using SSL or TLS proxy mode.

See the SSL/TLS support section of the *IBM MQ Internet Pass-Thru* documentation for more information.

For more information about connecting a queue manager using one-way authentication, that is, when the SSL or TLS client does not send a certificate, see Connecting two queue managers using one-way authentication.

IBM i, UNIX, Linux, and Windows systems

IBM i UNIX Linux Windows The SSL or TLS server sends a certificate to the client.

For queue managers and clients respectively, the following sources are searched in sequence for a non-empty value. The first non-empty value determines the certificate label. The certificate label must exist in the key repository. If no matching certificate in the correct case and format is found that matches a label, an error occurs and the SSL or TLS handshake fails.

Queue managers

1. Channel certificate label attribute **CERTLABL**.
2. Queue manager certificate label attribute **CERTLABL**.
3. A default, which is in the format: `ibmwebspheremq` with the name of the queue manager appended, all in lowercase. For example, for a queue manager named `QM1`, the default certificate label is `ibmwebspheremqqm1`.

IBM MQ clients

1. Channel certificate label attribute, **CERTLABL**.
2. MQSCO structure **CertificateLabel** attribute.
3. Environment variable **MQCERTLABL**.
4. Client `.ini` file (in its SSL section) **CertificateLabel** attribute
5. A default, which is in the format: `ibmwebspheremq` with the user ID that the client application is running as appended, all in lowercase. For example, for a user ID of `USER1`, the default certificate label is `ibmwebspheremquser1`.

z/OS

z/OS

z/OS IBM MQ Clients are not supported on z/OS systems. However, a z/OS queue manager can act in the role of an SSL or TLS client when initiating a connection, or an SSL or TLS server when accepting a connection request. Certificate label requirements for z/OS queue managers apply in both of these roles, and differ from the requirements on distributed platforms.

For queue managers and clients respectively, the following sources are searched in sequence for a non-empty value. The first non-empty value determines the certificate label. The certificate label must exist in the key repository. If no matching certificate in the correct case and format is found that matches a label, an error occurs and the SSL or TLS handshake fails.

1. Channel certificate label attribute, **CERTLABL**.
2. If shared, the queue-sharing group certificate label attribute, **CERTQSGL**.

   If not shared, the queue manager certificate label attribute, **CERTLABL**.
3. A default, which is in the format: `ibmWebSphereMQ` with the name of the queue manager or queue sharing group appended. Note that this string is case-sensitive and must be written as shown. For example, for a queue manager named `QM1`, the default certificate label is `ibmWebSphereMQQM1`.

For information on how to display the key repository, see "Locating the key repository for a queue manager on z/OS" on page 705.

IBM MQ Java and IBM MQ JMS clients

IBM MQ Java and IBM MQ JMS clients use the facilities of their Java Secure Socket Extension (JSSE) provider to select a personal certificate during the SSL or TLS handshake and are not therefore subject to certificate label requirements.

The default behavior is that the JSSE client iterates through the certificates in the key repository, selecting the first acceptable personal certificate found. However, this behavior is only a default, and is dependent on the implementation of the JSSE provider.

In addition, the JSSE interface is highly customizable through configuration and direct access at runtime by the application. Consult the documentation supplied by your JSSE provider for specific details.

For troubleshooting, or to better understand the handshake performed by the IBM MQ Java client application in combination with your specific JSSE provider, you can enable debugging by setting `javax.net.debug=ssl` in the JVM environment.

You can set the variable within the application, through configuration, or by entering `-Djavax.net.debug=ssl` on the command line.

IBM MQ client for HP Integrity NonStop Server

IBM MQ client for HP Integrity NonStop Server uses the channel **CERTLABL** attribute (or the value of the **MQCERTLABL** environment variable) to choose a certificate to be used for an SSL connection.

The certificate label, whether derived from the channel's **CERTLABL** attribute or from the value of the **MQCERTLABL** environment variable, is used to construct the name of the cert*.pem file and the corresponding Stash*.sth file.

```
cert_<CERTLABL>.pem
Stash_<CERTLABL>.sth
```

If the channel does not have a **CERTLABL** attribute and the **MQCERTLABL** environment variable is not present, then IBM MQ uses the following certificate and stash files:

```
cert_default_ <userid>.pem
Stash_default_ <userid>.sth
```

where <*userid* > is the name of the user that is executing the IBM MQ client program.

If the cert_default_<userid>.pem file or the Stash_default_<userid>.sth file does not exist, then IBM MQ uses the following certificate and stash files:

```
cert.pem
Stash.sth
```

Note:

1. The fallback to the cert_default_<userid>.pem and Stash_default_<userid>.sth files or the unqualified cert.pem and Stash.sth files only occurs if a certificate label is available from neither the channel's **CERTLABL** attribute nor the **MQCERTLABL** environment variable. If a **CERTLABL** is specified, then the associated cert_< *label* >.pem and Stash_ <*label*>.sth files must exist and be accessible for the connection to proceed.
2. IBM MQ rejects certificate labels containing any of the following characters:

   /

   \

   ⌂

   $

   and also rejects certificate labels containing more than one consecutive dot (.)

Refreshing the queue manager's key repository:

When you change the contents of a key repository, the queue manager does not immediately pick up the new contents. For a queue manager to use the new key repository contents, you must issue the REFRESH SECURITY TYPE(SSL) command.

This process is intentional, and prevents the situation where multiple running channels could use different versions of a key repository. As a security control, only one version of a key repository can be loaded by the queue manager at any time.

For more information about the REFRESH SECURITY TYPE(SSL) command, see REFRESH SECURITY.

You can also refresh a key repository using PCF commands or the IBM MQ Explorer. For more information, see the MQCMD_REFRESH_SECURITY command and the topic *Refreshing SSL or TLS Security* in the IBM MQ Explorer section of this product documentation.

Related concepts:
"Refreshing a client's view of the SSL key repository contents and SSL settings"
To update the client application with the refreshed contents of the key repository, you must stop and restart the client application.

Refreshing a client's view of the SSL key repository contents and SSL settings:

To update the client application with the refreshed contents of the key repository, you must stop and restart the client application.

You cannot refresh security on an IBM MQ client; there is no equivalent of the REFRESH SECURITY TYPE(SSL) command for clients (see REFRESH SECURITY) for more information.

You must stop and restart the application, whenever you change the security certificate, to update the client application with the refreshed contents of the key repository.

If restarting the channel refreshes the configurations, and if your application has reconnection logic, it is possible for you to refresh security at the client by issuing the STOP CHL STATUS(INACTIVE) command.
Related concepts:
"Refreshing the queue manager's key repository" on page 403
When you change the contents of a key repository, the queue manager does not immediately pick up the new contents. For a queue manager to use the new key repository contents, you must issue the REFRESH SECURITY TYPE(SSL) command.

MQCSP password protection:

From IBM MQ Version 8.0 , you can send passwords that are included in the MQCSP structure either protected, by using IBM MQ functionality, or encrypted, by using SSL/TLS encryption.

MQCSP password protection is useful for test and development purposes as using MQCSP password protection is simpler than setting up TLS encryption, but not as secure. For production purposes, you should use TLS encryption in preference to IBM MQ password protection, especially when the network between the client and queue manager is untrusted, as TLS encryption is more secure.

If you are concerned precisely what encryption is being used, and how much protection it offers, you need to use full TLS encryption. In this situation, the algorithms are publicly known, and you can select the appropriate one for your enterprise by using the **SSLCIPH** channel attribute.

For more information about the MQCSP structure, see MQCSP structure.

Password protection is used when all of the following conditions are met:
- Both ends of the connection are using IBM MQ Version 8.0, or later.
- The channel is not using TLS encryption. A channel is not using TLS encryption if the channel has a blank **SSLCIPH** attribute, or the **SSLCIPH** attribute is set to a CipherSpec that does not provide encryption. Null ciphers, for example, NULL_SHA, do not provide encryption.
- You set **MQCSP**.**AuthenticationType** to MQCSP_AUTH_USER_ID_AND_PWD. Setting this value enables more checks to be evaluated to decide whether password protection is done. The default value of **MQCSP**.**AuthenticationType** is MQCSP_AUTH_NONE. With the default setting, no password protection is done. For more information, see **AuthenticationType**.
- If the client is IBM MQ Explorer and user identification compatibility mode is not enabled, which is not the default. This condition is applicable only to IBM MQ Explorer.

If these conditions are not met, the password is sent in plain text unless prohibited by the **PasswordProtection** configuration setting.

The `PasswordProtection` configuration setting

The **`PasswordProtection`** attribute in the Channels section of the client and queue manager .ini configuration files can prevent passwords from being sent in plain text. The attribute can take 1 of 3 values. The default value is `compatible`:

compatible

The password can be sent in plain text if either the queue manager or client is running a version of IBM MQ earlier than Version 8.0. That is, plain text passwords are allowed for compatibility.

Therefore:

- The password is sent encrypted by the TLS CipherSpec if TLS encryption is used and the CipherSpec is not null.
- The password is sent in plain text if either the queue manager or the client is running a version of IBM MQ earlier than Version 8.0, and TLS encryption is not used. The password is sent in plain text as versions of IBM MQ earlier than Version 8.0 can send passwords only in plain text.
- The password is sent protected if both the queue manager and the client are running a version of IBM MQ at Version 8.0 or later, and either a null CipherSpec is used, or TLS encryption is not used. **`MQCSP`**.**`AuthenticationType`** must be set to MQCSP_AUTH_USER_ID_AND_PWD.
- The connection fails before the password is sent if both the queue manager and the client are running a version of IBM MQ at Version 8.0 or later, and **`MQCSP`**.**`AuthenticationType`** is not set to MQCSP_AUTH_USER_ID_AND_PWD.

always

The password must be either encrypted with a CipherSpec that is not a null CipherSpec, or **`MQCSP`**.**`AuthenticationType`** must be set to MQCSP_AUTH_USER_ID_AND_PWD. Otherwise, the connection fails. That is, plain text passwords are not allowed.

Therefore:

- The password is sent encrypted by the TLS CipherSpec if TLS encryption is used and the CipherSpec is not null.
- The password is sent protected if both the queue manager and the client are running a version of IBM MQ at Version 8.0 or later, and either TLS encryption is not used, or a null CipherSpec is used. **`MQCSP`**.**`AuthenticationType`** must be set to MQCSP_AUTH_USER_ID_AND_PWD.
- The connection fails before the password is sent if either the queue manager or the client is running a version of IBM MQ earlier than Version 8.0, and TLS encryption is not used. As versions of IBM MQ earlier than Version 8.0 can send passwords only in plain text, and `always` requires the password to be either encrypted or protected, the connection fails.

optional

The password can optionally be sent protected, but is sent in plain text if **`MQCSP`**.**`AuthenticationType`** is not set to MQCSP_AUTH_USER_ID_AND_PWD. That is, plain text passwords are allowed to be sent by any client.

Therefore:

- The password is sent encrypted by the TLS CipherSpec if TLS encryption is used and the CipherSpec is not null.
- The password is sent in plain text if a null CipherSpec is used and **`MQCSP`**.**`AuthenticationType`** is not set to MQCSP_AUTH_USER_ID_AND_PWD.
- The password is sent in plain text if either the queue manager or the client is running a version of IBM MQ earlier than Version 8.0, and TLS encryption is not used. The password is sent in plain text as versions of IBM MQ earlier than Version 8.0 can send passwords only in plain text.

- The password is sent protected if both the queue manager and the client are running a version of IBM MQ at Version 8.0 or later, TLS encryption is not used or a null CipherSpec is used, and **MQCSP**.**AuthenticationType** is set to MQCSP_AUTH_USER_ID_AND_PWD.

For Java and JMS clients, the behavior of the **PasswordProtection** attribute changes dependent on the choice of using compatibility mode or MQCSP mode:

- If Java and JMS clients are operating in compatibility mode, an MQCSP structure is not flowed during connection processing. Therefore, the behavior of the **PasswordProtection** attribute is the same behavior as described for clients that are running a version of IBM MQ earlier than Version 8.0.
- If Java and JMS clients are operating in MQCSP mode, the behavior of the **PasswordProtection** attribute is the behavior as described.

For more information about connection authentication with Java and JMS clients, see "Connection authentication with the Java client" on page 442.

Digital Certificate Manager (DCM):

Use the DCM to manage digital certificates and private keys on IBM i.

The Digital Certificate Manager (DCM) enables you to manage digital certificates and to use them in secure applications on the IBM i server. With Digital Certificate Manager, you can request and process digital certificates from Certificate Authorities (CAs) or other third-parties. You can also act as a local Certificate Authority to create and manage digital certificates for your users.

DCM also supports using Certificate Revocation Lists (CRLs) to provide a stronger certificate and application validation process. You can use DCM to define the location where a specific Certificate Authority CRL resides on an LDAP server so that IBM MQ can verify that a specific certificate has not been revoked.

DCM supports and can automatically detect certificates in a variety of formats. When DCM detects a PKCS #12 encoded certificate, or a PKCS #7 certificate that contains encrypted data, it automatically prompts the user to enter the password that was used to encrypt the certificate. DCM does not prompt for PKCS #7 certificates that do not contain encrypted data.

DCM provides a browser-based user interface that you can use to manage digital certificates for your applications and users. The user interface is divided into two main frames: a navigation frame and a task frame.

You use the navigation frame to select the tasks to manage certificates or the applications that use them. Some individual tasks are shown directly in the main navigation frame, but most tasks in the navigation frame are organized into categories. For example, Manage Certificates is a task category that contains various individual guided tasks, such as View certificate, Renew certificate, and Import certificate. If an item in the navigation frame is a category that contains more than one task, an arrow is displayed to the left of it. The arrow indicates that when you select the category link, an expanded list of tasks displays, enabling you to choose which task to perform.

For important information about DCM, see the following IBM Redbooks® publications:

- *IBM i Wired Network Security: OS/400 V5R1 DCM and Cryptographic Enhancements*, SG24-6168. Specifically, see the appendixes for essential information about setting up your IBM i system as a local CA.
- *AS/400 Internet Security: Developing a Digital Certificate Infrastructure*, SG24-5659. Specifically, see Chapter 5. *Digital Certificate Manager for AS/400* , which explains the AS/400 DCM.

Federal Information Processing Standards (FIPS):

This topic introduces the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology and the cryptographic functions which can be used on SSL or TLS channels.

This information applies to the following platforms:
- Windows
- UNIX and Linux
- z/OS

The FIPS 140-2 compliance of an IBM MQ SSL or TLS connection on UNIX, Linux, and Windows systems is found here "Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows."

z/OS The FIPS 140-2 compliance of an IBM MQ SSL or TLS connection on z/OS is found here "Federal Information Processing Standards (FIPS) for z/OS" on page 410.

If cryptographic hardware is present, the cryptographic modules used by IBM MQ can be configured to be those provided by the hardware manufacturer. If this is done, the configuration is only FIPS-compliant if those cryptographic modules are FIPS-certified.

Over time, the Federal Information Processing Standards are updated to reflect new attacks against encryption algorithms and protocols. For example, some CipherSpecs may cease to be FIPS certified. When such changes occur, IBM MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance.

The IBM MQ product readme lists the version of FIPS enforced by each product maintenance level. If you configure IBM MQ to enforce FIPS compliance, always consult the readme when planning to apply maintenance. The readme is located at IBM MQ product READMEs

Related concepts:

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 653
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

"Using runmqckm, runmqakm, and strmqikm to manage digital certificates" on page 673
On UNIX, Linux and Windows systems, manage keys and digital certificates with the **`strmqikm`** (iKeyman) GUI, or from the command line using **`runmqckm`** (iKeycmd) or **`runmqakm`** (GSKCapiCmd).

Related information:

Enabling SSL in IBM MQ classes for Java

SSL properties of JMS objects

Using Secure Sockets Layer (SSL) with IBM MQ classes for JMS

"Federal Information Processing Standards" on page 394
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows:

When cryptography is required on an SSL or TLS channel on Windows, UNIX and Linux systems, IBM MQ uses a cryptography package called IBM Crypto for C (ICC). On the Windows, UNIX and Linux platforms, the ICC software has passed the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2.

The FIPS 140-2 compliance of an IBM MQ SSL or TLS connection on Windows, UNIX and Linux systems is as follows:

- For all IBM MQ message channels (except CLNTCONN channel types), the connection is FIPS-compliant if the following conditions are met:
  - The installed GSKit ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - The queue manager's SSLFIPS attribute has been set to YES.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
- For all IBM MQ MQI client applications , the connection uses GSKit and is FIPS-compliant if the following conditions are met:
  - The installed GSKit ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the MQI client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
- For IBM MQ classes for Java applications using client mode, the connection uses the JRE's SSL and TLS implementations and is FIPS-compliant if the following conditions are met:
  - The Java Runtime Environment used to run the application is FIPS-compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the Java client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
- For IBM MQ classes for JMS applications using client mode, the connection uses the JRE's SSL and TLS implementations and is FIPS-compliant if the following conditions are met:
  - The Java Runtime Environment used to run the application is FIPS-compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the JMS client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
- For unmanaged .NET client applications, the connection uses GSKit and is FIPS-compliant if the following conditions are met:
  - The installed GSKit ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the .NET client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
- For unmanaged XMS .NET client applications, the connection uses GSKit and is FIPS-compliant if the following conditions are met:
  - The installed GSKit ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the XMS .NET documentation.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.

All supported platforms are FIPS 140-2 certified except as noted in the readme file included with each fix pack or refresh pack.

For SSL and TLS connections using GSKit, the component which is FIPS 140-2 certified is named *ICC*. It is the version of this component which determines GSKit FIPS compliance on any given platform. To determine the ICC version currently installed, run the **dspmqver -p 64 -v** command.

Here is an example extract of the **dspmqver -p 64 -v** output relating to ICC:

```
ICC
============
@(#)CompanyName:   IBM Corporation
@(#)LegalTrademarks: IBM
@(#)FileDescription: IBM Crypto for C-language
@(#)FileVersion:   8.0.0.0
@(#)LegalCopyright:  Licensed Materials - Property of IBM
@(#)         ICC
@(#)         (C) Copyright IBM Corp. 2002,2010
@(#)         All Rights Reserved. US Government Users
@(#)         Restricted Rights - Use, duplication or disclosure
@(#)         restricted by GSA ADP Schedule Contract with IBM Corp.
@(#)ProductName:   icc_8.0 (GoldCoast Build) 100415
@(#)ProductVersion:  8.0.0.0
@(#)ProductInfo:   10/04/15.03:32:19.10/04/15.18:41:51
@(#)CMVCInfo:
```

The NIST certification statement for GSKit ICC 8 (included in GSKit 8) can be found at the following address: http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401val2013.htm#1994.

If cryptographic hardware is present, the cryptographic modules used by IBM MQ can be configured to be those provided by the hardware manufacturer. If this is done, the configuration is only FIPS-compliant if those cryptographic modules are FIPS-certified.

Note: 32 bit Solaris x86 SSL and TLS clients configured for FIPS 140-2 compliant operation fail when running on Intel systems. This failure occurs because the FIPS 140-2 compliant GSKit-Crypto Solaris x86 32 bit library file does not load on the Intel chipset. On affected systems, error AMQ9655 is reported in the client error log. To resolve this issue, disable FIPS 140-2 compliance or recompile the client application 64 bit, because 64 bit code is not affected.

Triple DES restrictions enforced when operating in compliance with FIPS 140-2

When IBM MQ is configured to operate in compliance with FIPS 140-2, additional restrictions are enforced in relation to Triple DES (3DES) CipherSpecs. These restrictions enable compliance with the US NIST SP800-67 recommendation.

1. All parts of the Triple DES key must be unique.
2. No part of the Triple DES key can be a Weak, Semi-Weak, or Possibly-Weak key according to the definitions in NIST SP800-67.
3. No more than 32 GB of data can be transmitted over the connection before a secret key reset must occur. By default, IBM MQ does not reset the secret session key so this reset must be configured. Failure to enable secret key reset when using a Triple DES CipherSpec and FIPS 140-2 compliance results in the connection closing with error AMQ9288 after the maximum byte count is exceeded. For information about how to configure secret key reset, see "Resetting SSL and TLS secret keys" on page 804.

IBM MQ generates Triple DES session keys which already comply with rules 1 and 2. However, to satisfy the third restriction you must enable secret key reset when using Triple DES CipherSpecs in a FIPS 140-2 configuration. Alternatively, you can avoid using Triple DES.

Related concepts:

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 653
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

"Using runmqckm, runmqakm, and strmqikm to manage digital certificates" on page 673
On UNIX, Linux and Windows systems, manage keys and digital certificates with the **strmqikm** (iKeyman) GUI, or from the command line using **runmqckm** (iKeycmd) or **runmqakm** (GSKCapiCmd).

Related information:

Enabling SSL in IBM MQ classes for Java

SSL properties of JMS objects

Using Secure Sockets Layer (SSL) with IBM MQ classes for JMS

"Federal Information Processing Standards" on page 394
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

Federal Information Processing Standards (FIPS) for z/OS:

When cryptography is required on an SSL or TLS channel on z/OS , IBM MQ uses a service called System SSL. The objective of System SSL is to provide the capability to execute securely in a mode designed to adhere to the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2.

When implementing FIPS 140-2 compliant connections with IBM MQ SSL or TLS connections there are a number of points to consider:

- To enable IBM MQ message channels for FIPS-compliance, ensure the following conditions are met:
  - System SSL Security Level 3 FMID is installed and configured (see Planning to install IBM MQ).
  - System SSL modules are validated.
  - The queue manager's SSLFIPS attribute has been set to **YES**.

When executing in FIPS mode, System SSL exploits CP Assist for Cryptographic Function (CPACF) when available. Cryptographic functions performed by ICSF-supported hardware when running in non-FIPS mode continue to be exploited when executing in FIPS mode, with the exception of RSA signature generation which must be performed in software.

Table 16. Differences between FIPS mode and non-FIPS mode algorithm support.

	Non-FIPS		**FIPS**	
Algorithm	**Key sizes**	**Hardware**	**Key sizes**	**Hardware**
RC2	40 and 128			
RC4	40 and 128			
DES	56	x		
TDES	168	x	168	x
AES	128 and 256	x	128 and 256	x
MD5	48			
SHA-1	160	x	160	x
SHA-2	224, 256, 384 and 512	x	224, 256, 384 and 512	x
RSA	512-4096	x	1024-4096	x
DSA	512-1024		1024	

Table 16. Differences between FIPS mode and non-FIPS mode algorithm support. (continued)

	Non-FIPS		FIPS	
Algorithm	**Key sizes**	**Hardware**	**Key sizes**	**Hardware**
DH	512-2048		2048	

In FIPS mode, System SSL can only use certificates that use the algorithms and key sizes shown in Table 1. During X.509 certificate validation if an algorithm that is incompatible with FIPS mode is encountered, then the certificate cannot be used and is treated as not valid.

For IBM MQ classes applications using client mode within WebSphere Application Server , refer to the Federal Information Processing Standards-approved section of the IBM WebSphere Application Server Version 7 product documentation.

For information on System SSL module configuration, see System SSL Module Verification Setup.

Related information:

"Federal Information Processing Standards" on page 394
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

SSL and TLS on the IBM MQ MQI client:

IBM MQ supports SSL and TLS on clients. You can tailor the use of SSL or TLS in various ways.

IBM MQ provides SSL and TLS support for IBM MQ MQI clients on Windows, UNIX and Linux systems. If you are using IBM MQ classes for Java, see Using IBM MQ classes for Java and if you are using IBM MQ classes for JMS, see Using IBM MQ classes for JMS. The rest of this section does not apply to the Java or JMS environments.

You can specify the key repository for an IBM MQ MQI client either with the MQSSLKEYR value in your IBM MQ client configuration file, or when your application makes an MQCONNX call. You have three options for specifying that a channel uses SSL:

- Using a channel definition table
- Using the SSL configuration options structure, MQSCO, on an MQCONNX call
- Using the Active Directory (on Windows systems)

You cannot use the MQSERVER environment variable to specify that a channel uses SSL.

You can continue to run your existing IBM MQ MQI client applications without SSL, as long as SSL is not specified at the other end of the channel.

If changes are made on a client machine to the contents of the SSL Key Repository, the location of the SSL Key Repository, the Authentication Information, or the Cryptographic hardware parameters, you need to end all the SSL connections in order to reflect these changes in the client-connection channels that the application is using to connect to the queue manager. Once all the connections have ended, restart the SSL channels. All the new SSL settings are used. These settings are analogous to those refreshed by the REFRESH SECURITY TYPE(SSL) command on queue manager systems.

When your IBM MQ MQI client runs on a Windows, UNIX and Linux system with cryptographic hardware, you configure that hardware with the MQSSLCRYP environment variable. This variable is equivalent to the SSLCRYP parameter on the ALTER QMGR MQSC command. Refer to ALTER QMGR for a description of the SSLCRYP parameter on the ALTER QMGR MQSC command. If you use the GSK_PCS11 version of the SSLCRYP parameter, the PKCS #11 token label must be specified entirely in lower-case.

SSL secret key reset and FIPS are supported on IBM MQ MQI clients. For more information, see "Resetting SSL and TLS secret keys" on page 804 and "Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows" on page 407.

See "Setting up IBM MQ MQI client security" on page 652 for more information about the SSL support for IBM MQ MQI clients.

Related information:

Configuring a client using a configuration file

Specifying that an MQI channel uses SSL:

For an MQI channel to use SSL, the value of the *SSLCipherSpec* attribute of the client-connection channel must be the name of a CipherSpec that is supported by IBM MQ on the client platform.

You can define a client-connection channel with a value for this attribute in the following ways. They are listed in order of decreasing precedence.

1. When a PreConnect exit provides a channel definition structure to use.

   A PreConnect exit can provide the name of a CipherSpec in the *SSLCipherSpec* field of a channel definition structure, MQCD. This structure is returned in the **ppMQCDArrayPtr** field of the MQNXP exit parameter structure used by the PreConnect exit.
2. When an IBM MQ MQI client application issues an MQCONNX call.

   The application can specify the name of a CipherSpec in the *SSLCipherSpec* field of a channel definition structure, MQCD. This structure is referenced by the connect options structure, MQCNO, which is a parameter on the MQCONNX call.
3. Using a client channel definition table (CCDT).

   One or more entries in a client channel definition table can specify the name of a CipherSpec. For example, if you create an entry by using the DEFINE CHANNEL MQSC command, you can use the SSLCIPH parameter on the command to specify the name of a CipherSpec.
4. Using Active Directory on Windows.

   On Windows systems, you can use the **setmqscp** control command to publish the client-connection channel definitions in Active Directory. One or more of these definitions can specify the name of a CipherSpec.

For example, if a client application provides a client-connection channel definition in an MQCD structure on an MQCONNX call, this definition is used in preference to any entries in a client channel definition table that can be accessed by the IBM MQ client.

You cannot use the MQSERVER environment variable to provide the channel definition at the client end of an MQI channel that uses SSL.

To check whether a client certificate has flowed, display the channel status at the server end of a channel for the presence of a peer name parameter value.

Related concepts:

"Specifying a CipherSpec for an IBM MQ MQI client" on page 803
You have three options for specifying a CipherSpec for an IBM MQ MQI client.

CipherSpecs and CipherSuites in IBM MQ:

From IBM MQ Version 8.0.0, Fix Pack 2 SSL CipherSpecs are deprecated. TLS CipherSpecs are supported, and RSA and Diffie-Hellman algorithms.

IBM MQ supports TLS V1.0 and V1.2 CipherSpecs.

From IBM MQ Version 8.0.0, Fix Pack 2 the SSLv3 protocol and the use of some IBM MQ CipherSpecs is deprecated. For more information, see Deprecation: SSLv3 protocol.

IBM MQ supports the RSA and Diffie-Hellman key exchange and authentication algorithms. The size of the key used during the TSL, or SSL handshake can depend on the digital certificate you use, but some CipherSpecs include a specification of the handshake key size. Larger handshake key sizes provide stronger authentication. With smaller key sizes, the handshake is faster.

Related concepts:

"CipherSpecs and CipherSuites" on page 392
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

CipherSpec values supported in IBM MQ:

The set of default CipherSpecs allows only the following values:

TLS 1.0

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA

TLS 1.2

- ECDHE_ECDSA_AES_128_CBC_SHA256
- ECDHE_ECDSA_AES_256_CBC_SHA384
- ECDHE_ECDSA_AES_128_GCM_SHA256
- ECDHE_ECDSA_AES_256_GCM_SHA384
- ECDHE_ECDSA_3DES_EDE_CBC_SHA256
- ECDHE_RSA_AES_128_CBC_SHA256
- ECDHE_RSA_AES_256_CBC_SHA384
- ECDHE_RSA_AES_128_GCM_SHA256
- ECDHE_RSA_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384

IBM i The default set of CipherSpecs for IBM MQ for IBM i Version 7.2 and Version 7.3 allows only the following values:

- *TLS_RSA_WITH_AES_128_GCM_SHA256
- *TLS_RSA_WITH_AES_256_GCM_SHA384
- *ECDHE_ECDSA_RC4_128_SHA256

- *ECDHE_ECDSA_3DES_EDE_CBC_SHA256
- *ECDHE_RSA_RC4_128_SHA256
- *ECDHE_RSA_3DES_EDE_CBC_SHA256
- *ECDHE_ECDSA_AES_128_CBC_SHA256
- *ECDHE_ECDSA_AES_256_CBC_SHA384
- *ECDHE_RSA_AES_128_CBC_SHA256
- *ECDHE_RSA_AES_256_CBC_SHA384
- *ECDHE_ECDSA_AES_128_GCM_SHA256
- *ECDHE_ECDSA_AES_256_GCM_SHA384
- *ECDHE_RSA_AES_128_GCM_SHA256
- *ECDHE_RSA_AES_256_GCM_SHA384
- *ECDHE_RSA_NULL_SHA256
- *ECDHE_ECDSA_NULL_SHA256

Enabling deprecated TLS CipherSpecs

By default, you are not allowed to specify a deprecated CipherSpec on a channel definition. If you attempt to specify a deprecated CipherSpec, you receive message AMQ8242: SSLCIPH definition wrong, and PCF returns MQRCCF_SSL_CIPHER_SPEC_ERROR.

You cannot start a channel with a deprecated CipherSpec. If you attempt to do so with a deprecated CipherSpec, the system returns *MQCC_FAILED (2)*, together with a **Reason** of *MQRC_SSL_INITIALIZATION_ERROR (2393)* to the client.

It is possible for you to re-enable one or more of the deprecated CipherSpecs for defining channels, at runtime on the server, by setting the environment variable *AMQ_SSL_WEAK_CIPHER_ENABLE*.

The *AMQ_SSL_WEAK_CIPHER_ENABLE* environment variable accepts:
- A single CipherSpec name, or
- A comma separated list of IBM MQ CipherSpec names to re-enable, or
- The special value of *ALL*, representing all CipherSpecs.

For example, if you want to re-enable ECDHE_RSA_RC4_128_SHA256, set the following environment variable:

```
AMQ_SSL_WEAK_CIPHER_ENABLE=ECDHE_RA_RC4_128_SHA256
```

or, alternatively change the SSL stanza in the `qm.ini` file, by setting:

```
SSL
AllowWeakCipherSpec=ECDHE_RA_RC4_128_SHA256
```

Enabling deprecated SSL CipherSpecs

In addition to issuing `AMQ_SSL_WEAK_CIPHER_ENABLE`, or `AllowWeakCipherSpec`, as described in "Enabling deprecated TLS CipherSpecs," you must set the environment variable `AMQ_SSL_V3_ENABLE=1` or issue `AllowSSLV3=Y` as described in Deprecation:SSLV3 protocol.

For example, if you want to re-enable RC4_MD5_US, set the following environment variables:

```
AMQ_SSL_V3_ENABLE=1
AMQ_SSL_WEAK_CIPHER_ENABLE=RC4_MD5_US
```

or, alternatively, change the SSL stanza in the `qm.ini` file, by setting:

```
SSL
AllowSSLV3=Y
AllowWeakCipherSpec=RC4_MD5_US
```

Related concepts:

"CipherSpecs and CipherSuites" on page 392
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

NSA Suite B Cryptography in IBM MQ:

This topic provides information about how to configure IBM MQ on Windows, Linux, and UNIX systems to conform to the Suite B compliant TLS 1.2 profile.

Over time, the NSA Cryptography Suite B Standard is updated to reflect new attacks against encryption algorithms and protocols. For example, some CipherSpecs might cease to be Suite B certified. When such changes occur, IBM MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance. The IBM MQ 7.5 readme file lists the version of Suite B enforced by each product maintenance level. If you configure IBM MQ to enforce Suite B compliance, always consult the readme file when planning to apply maintenance. The readme file is at IBM MQ product READMEs.

On Windows, UNIX, and Linux systems, IBM MQ can be configured to conform to the Suite B compliant TLS 1.2 profile at the security levels shown in Table 1.

Table 17. Suite B security levels with allowed CipherSpecs and digital signature algorithms

Security level	Allowed CipherSpecs	Allowed digital signature algorithms
128-bit	ECDHE_ECDSA_AES_128_GCM_SHA256 ECDHE_ECDSA_AES_256_GCM_SHA384	ECDSA with SHA-256 ECDSA with SHA-384
192-bit	ECDHE_ECDSA_AES_256_GCM_SHA384	ECDSA with SHA-384
Both [1]	ECDHE_ECDSA_AES_128_GCM_SHA256 ECDHE_ECDSA_AES_256_GCM_SHA384	ECDSA with SHA-256 ECDSA with SHA-384

1. It is possible to configure both the 128-bit and 192-bit security levels concurrently. Since the Suite B configuration determines the minimum acceptable cryptographic algorithms, configuring both security levels is equivalent to configuring only the 128-bit security level. The cryptographic algorithms of the 192-bit security level are stronger than the minimum required for the 128-bit security level, so they are permitted for the 128-bit security level even if the 192-bit security level is not enabled.

Note: The naming conventions used for the Security level do not necessarily represent the elliptic curve size or the key size of the AES encryption algorithm.

CipherSpec conformation to Suite B

Although the default behavior of IBM MQ is not to comply with the Suite B standard, IBM MQ can be configured to conform to either, or both security levels on Windows, UNIX and Linux systems. Following the successful configuration of IBM MQ to use Suite B, any attempt to start an outbound channel using a CipherSpec not conforming to Suite B results in the error `AMQ9282`. This activity also results in the MQI client returning the reason code `MQRC_CIPHER_SPEC_NOT_SUITE_B`. Similarly, attempting to start an inbound channel using a CipherSpec not conforming to the Suite B configuration results in the error `AMQ9616`.

For more information about IBM MQ CipherSpecs, see "Enabling CipherSpecs" on page 797

Suite B and digital certificates

Suite B restricts the digital signature algorithms which can be used to sign digital certificates. Suite B also restricts the type of public key which certificates can contain. Therefore IBM MQ must be configured to use certificates whose digital signature algorithm and public key type are allowed by the configured Suite B security level of the remote partner. Digital certificates which do not comply with the security level requirements are rejected and the connection fails with error `AMQ9633` or `AMQ9285`.

For the 128-bit Suite B security level, the public key of the certificate subject is required to use either the NIST P-256 elliptic curve or the NIST P-384 elliptic curve and to be signed with either the NIST P-256 elliptic curve or the NIST P-384 elliptic curve. At the 192-bit Suite B security level, the public key of the certificate subject is required to use the NIST P-384 elliptic curve and to be signed with the NIST P-384 elliptic curve.

To obtain a certificate suitable for Suite B compliant operation, use the **`runmqakm`** command and specify the **`-sig_alg`** parameter to request a suitable digital signature algorithm. The `EC_ecdsa_with_SHA256` and `EC_ecdsa_with_SHA384` **`-sig_alg`** parameter values correspond to elliptic curve keys signed by the allowed Suite B digital signature algorithms.

For more information about the **`runmqakm`** command, see runmqckm and runmqakm options.

Note: The **`iKeycmd`** and **`iKeyman`** tools do not support the creation of digital certificates for Suite B compliant operation.

Creating and requesting digital certificates

To create a self-signed digital certificate for Suite B testing, see “Creating a self-signed personal certificate on UNIX, Linux, and Windows systems” on page 681

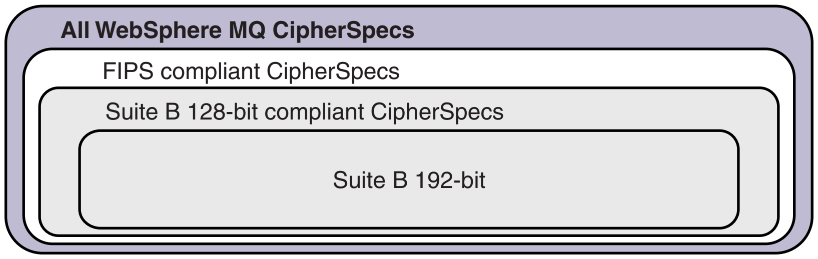
To request a CA-signed digital certificate for Suite B production use, see “Requesting a personal certificate on UNIX, Linux, and Windows systems” on page 683.

Note: The certificate authority being used must generate digital certificates which satisfy the requirements described in IETF RFC 6460.

FIPS 140-2 and Suite B

The Suite B standard is conceptually similar to FIPS 140-2, as it restricts the set of enabled cryptographic algorithms in order to provide an assured level of security. The Suite B CipherSpecs currently supported can be used when IBM MQ is configured for FIPS 140-2 compliant operation. It is therefore possible to configure IBM MQ for both FIPS and Suite B compliance simultaneously, in which case both sets of restrictions apply.

The following diagram illustrates the relationship between these subsets:


All WebSphere MQ CipherSpecs
FIPS compliant CipherSpecs
Suite B 128-bit compliant CipherSpecs
Suite B 192-bit

Configuring IBM MQ for Suite B compliant operation

For information about how to configure IBM MQ on Windows, UNIX and Linux for Suite B compliant operation, see "Configuring IBM MQ for Suite B."

IBM MQ does not support Suite B compliant operation on the IBM i and z/OS platforms. The IBM MQ Java and JMS clients also do not support Suite B compliant operation.

Related concepts:

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 653
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

Configuring IBM MQ for Suite B:

IBM MQ can be configured to operate in compliance with the NSA Suite B standard on Windows, UNIX and Linux platforms.

Suite B restricts the set of enabled cryptographic algorithms in order to provide an assured level of security. IBM MQ can be configured to operate in compliance with Suite B to provide an enhanced level of security. For further information on Suite B, see "National Security Agency (NSA) Suite B Cryptography" on page 394. For more information about Suite B configuration and its effect on SSL and TLS channels, see "NSA Suite B Cryptography in IBM MQ" on page 415.

Queue manager

For a queue manager, use the command **ALTER QMGR** with the parameter **SUITEB** to set the values appropriate for your required level of security. For further information see ALTER QMGR.

You can also use the PCF **MQCMD_CHANGE_Q_MGR** command with the **MQIA_SUITE_B_STRENGTH** parameter to configure the queue manager for Suite B compliant operation.

Note: If you alter a queue manager's Suite B settings, you must restart the MQXR service for those settings to take effect.

MQI client

By default, MQI clients do not enforce Suite B compliance. You can enable the MQI client for Suite B compliance by executing one of the following options:

1. By setting the **EncryptionPolicySuiteB** field in the MQSCO structure on an MQCONNX call to one or more of the following values:
   - MQ_SUITE_B_NONE
   - MQ_SUITE_B_128_BIT
   - MQ_SUITE_B_192_BIT

   Using MQ_SUITE_B_NONE with any other value is invalid.
2. By setting the MQSUITEB environment variable to one or more of the following values:
   - NONE
   - 128_BIT
   - 192_BIT

   You can specify multiple values using a comma separated list. Using the value NONE with any other value is invalid.
3. By setting the **EncryptionPolicySuiteB** attribute in the SSL stanza of the MQI client configuration file to one or more of the following values:
   - NONE

- 128_BIT
- 192_BIT

You can specify multiple values using a comma separated list. Using NONE with any other value is invalid.

Note: The MQI client settings are listed in order of priority. The MSCO structure on the MQCONNX call overrides the setting on the MQSUITEB environment variable, which overrides the attribute in the SSL stanza.

For full details of the MQSCO structure, see MQSCO - SSL configuration options.

For more information about the use of Suite B in the client configuration file, see SSL stanza of the client configuration file.

For further information on the use of the MQSUITEB environment variable, see Environment Variables.

.NET

For .NET unmanaged clients, the property **`MQC.ENCRYPTION_POLICY_SUITE_B`** indicates the type of Suite B security required.

For information about the using Suite B in IBM MQ classes for .NET, see MQEnvironment .NET class.

V 8.0.0.4

AMQP

The Suite B attribute settings for a queue manager apply to AMQP channels on that queue manager. If you modify the queue manager Suite B settings, you must restart the AMQP service for the changes to take effect.

Certificate validation policies in IBM MQ:

The certificate validation policy determines how strictly the certificate chain validation conforms to industry security standards.

The certificate validation policy depends upon the platform and environment as follows:
- For Java and JMS applications on all platforms, the certificate validation policy depends on the JSSE component of the Java runtime environment. For more information about the certificate validation policy, see the documentation for your JRE.
- For IBM i systems, the certificate validation policy depends on the secure sockets library provided by the operating system. For more information about the certificate validation policy, see the documentation for the operating system.
- For z/OS systems, the certificate validation policy depends on the System SSL component provided by the operating system. For more information about the certificate validation policy, see the documentation for the operating system.
- For UNIX, Linux, and Windows systems, the certificate validation policy is supplied by GSKit and can be configured. Two different certificate validation policies are supported:
  - A legacy certificate validation policy, used for maximum backwards compatibility and interoperability with old digital certificates that do not comply with the current IETF certificate validation standards. This policy is known as the Basic policy.
  - A strict, standards-compliant certificate validation policy which enforces the RFC 5280 standard. This policy is known as the Standard policy.

For information about how to configure the certificate validation policy on UNIX, Linux, and Windows systems, see "Configuring certificate validation policies in IBM MQ." For more information about the differences between the Basic and Standard certificate validation policies, see Certificate validation and trust policy design on UNIX, Linux and Windows systems.

Configuring certificate validation policies in IBM MQ:

You can specify which SSL/TLS certificate validation policy is used to validate digital certificates received from remote partner systems in four ways.

On the queue manager, the certificate validation policy can be set in the following ways:
- Using the queue manager attribute *CERTVPOL*. For more information about setting this attribute, see ALTER QMGR.

On the client, there are several methods that can be used to set the certificate validation policy. If more than one method is used to set the policy, the client uses the settings in the following priority order:
1. Using the *CertificateValPolicy* field in the client MQSCO structure. For more information about using this field, see MQSCO - SSL configuration options.
2. Using the client environment variable, *MQCERTVPOL*. For more information about using this variable, see MQCERTVPOL.
3. Using the client SSL stanza tuning parameter setting, *CertificateValPolicy*. For more information about using this setting, see SSL stanza of the client configuration file.

For more information about certificate validation policies, see "Certificate validation policies in IBM MQ" on page 418.

Digital certificates and CipherSpec compatibility in IBM MQ:

This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

In releases prior to IBM WebSphere MQ Version 7.1, all supported SSL and TLS CipherSpecs used the RSA algorithm for digital signatures and key agreement. All of the supported types of digital certificate were compatible with all of the supported CipherSpecs, so it was possible to change the CipherSpec for any channel without needing to change digital certificates.

In IBM WebSphere MQ Version 7.1 and later, only a subset of the supported CipherSpecs can be used with all of the supported types of digital certificate. It is therefore necessary to choose an appropriate CipherSpec for your digital certificate. Similarly, if your organization's security policy requires that you use a particular CipherSpec then you must obtain an appropriate digital certificate for that CipherSpec.

The MD5 digital signature algorithm and TLS 1.2

Digital certificates signed using the MD5 algorithm are rejected when the TLS 1.2 protocol is used. This is because the MD5 algorithm is now considered weak by many cryptographic analysts and its use is generally discouraged. If you wish to use newer CipherSpecs based on the TLS 1.2 protocol, ensure that the digital certificates do not use the MD5 algorithm in their digital signatures. Older CipherSpecs which use the SSL 3.0 and TLS 1.0 protocols are not subject to this restriction and can continue to use certificates with MD5 digital signatures.

To view the digital signature algorithm for a particular certificate, you can use the **runmqakm** command:

```
runmqakm -cert -details -db key.kdb -pw password -label cert_label
```

where *cert_label* is the certificate label of the digital signature algorithm to to display. See Digital certificate labels for details.

Note: Although the **iKeycmd (runmqckm)** tool and the **iKeyman (strmqikm)** GUI can be used to view a selection of digital signature algorithms, the **runmqakm** tool provides a wider range.

The execution of the **runmqakm** command produces output displaying the use of the signature algorithm specified:

```
Label : ibmwebspheremqexample
Key Size : 1024
Version : X509 V3
Serial : 4e4e93f1
Issuer : CN=Old Certificate Authority,OU=Test,O=Example,C=US
Subject : CN=Example Queue Manager,OU=Test,O=Example,C=US
Not Before : August 19, 2011 5:48:49 PM GMT+01:00
Not After : August 18, 2012 5:48:49 PM GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 98 5A 7A
    F0 18 21 EE E4 8A 6E DE C8 01 4B 3A 1E 41 90 3D
    CE 01 3F E6 32 30 6C 23 59 F0 FE 78 6D C2 80 EF
    BC 83 54 7A EB 60 80 62 6B F1 52 FE 51 9D C1 61
    80 A5 1C D4 F0 76 C7 15 6D 1F 0D 4D 31 3E DC C6
    A9 20 84 6E 14 A1 46 7D 4C F5 79 4D 37 54 0A 3B
    A9 74 ED E7 8B 0F 80 31 63 1A 0B 20 A5 99 EE 0A
    30 A6 B6 8F 03 97 F6 99 DB 6A 58 89 7F 27 34 DE
    55 08 29 D8 A9 6B 46 E6 02 17 C3 13 D3 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    09 4E 4F F2 1B CB C1 F4 4F 15 C9 2A F7 32 0A 82
    DA 45 92 9F
Fingerprint : MD5 :
    44 54 81 7C 58 68 08 3A 5D 75 96 40 D5 8C 7A CB
Fingerprint : SHA256 :
    3B 47 C6 E7 7B B0 FF 85 34 E7 48 BE 11 F2 D4 35
    B7 9A 79 53 2B 07 F5 E7 65 E8 F7 84 E0 2E 82 55
Signature Algorithm : MD5WithRSASignature (1.2.840.113549.1.1.4)
Value
    3B B9 56 E6 F2 77 94 69 5B 3F 17 EA 7B 19 D0 A2
    D7 10 38 F1 88 A4 44 1B 92 35 6F 3B ED 99 9B 3A
    A5 A4 FC 72 25 5A A9 E3 B1 96 88 FC 1E 9F 9B F1
    C5 E8 8E CF C4 8F 48 7B 0E A6 BB 13 AE 2B BD D8
    63 2C 03 38 EF DC 01 E1 1F 7A 6F FB 2F 65 74 D0
    FD 99 94 BA B2 3A D5 B4 89 6C C1 2B 43 6D E2 39
    66 6A 65 CB C3 C4 E2 CC F5 49 39 A3 8B 93 5A DD
    B0 21 0B A8 B2 59 5B 24 59 50 44 89 DC 78 19 51
Trust Status : Enabled
```

The `Signature Algorithm` line shows that the `MD5WithRSASignature` algorithm is used. This algorithm is based upon MD5 and so this digital certificate cannot be used with the TLS 1.2 CipherSpecs.

Interoperability of Elliptic Curve and RSA CipherSpecs

Not all CipherSpecs can be used with all digital certificates. There are three types of CipherSpec, denoted by the CipherSpec name prefix. Each type of CipherSpec imposes different restrictions upon the type of digital certificate which may be used. These restrictions apply to all IBM MQ SSL and TLS connections, but are particularly relevant to users of Elliptic Curve cryptography.

The following table summarizes the relationships between CipherSpecs and digital certificates:

Table 18. Relationships between CipherSpecs and digital certificates

Type	CipherSpec Name Prefix	Description	Required public key type	Digital signature encryptionestablishment algorithm	Secret key establishment method
1	ECDHE_ECDSA_	CipherSpecs which use Elliptic Curve public keys, Elliptic Curve secret keys, and Elliptic Curve digital signature algorithms.	Elliptic Curve	ECDSA	ECDHE
2	ECDHE_RSA_	CipherSpecs which use RSA public keys, Elliptic Curve secret keys, and Elliptic Curve digital signature algorithms.	RSA	RSA	ECDHE
3	(All others)	CipherSpecs which use RSA public keys and RSA digital signature algorithms.	RSA	RSA	RSA

Note: Type 1 and 2 CipherSpecs are not supported by IBM MQ queue managers and MQI clients on the IBM i platform.

The required public key type column shows the type of public key which the personal certificate must have when using each type of CipherSpec. The personal certificate is the end-entity certificate which identifies the queue manager or client to its remote partner.

You could configure a channel with both a CipherSpec that requires an Elliptic Curve (EC) certificate, and a certificate label for an RSA certificate, or the other way round. You must ensure that the certificate named in the certificate label is appropriate for the channel CipherSpec.

Assuming that you have correctly configured IBM MQ, you can have:
- A single queue manager with a mixture of RSA and EC certificates.
- Different channels on the same queue manager using either an RSA or EC certificate.

The digital signature encryption algorithm refers to the encryption algorithm used to validate the peer. The encryption algorithm is used along with a hash algorithm such as MD5, SHA-1 or SHA-256 to compute the digital signature. There are various digital signature algorithms which can be used, for example "RSA with MD5" or "ECDSA with SHA-256". In the table, ECDSA refers to the set of digital signature algorithms which use ECDSA; RSA refers to the set of digital signature algorithms which use RSA. Any supported digital signature algorithm in the set may be used, provided it is based upon the stated encryption algorithm.

Type 1 CipherSpecs require that the personal certificate must have an Elliptic Curve public key. When these CipherSpecs are used, Elliptic Curve Diffie Hellman Ephemeral key agreement is used to establish the secret key for the connection.

Type 2 CipherSpecs require that the personal certificate has an RSA public key. When these CipherSpecs are used, Elliptic Curve Diffie Hellman Ephemeral key agreement is used to establish the secret key for the connection.

Type 3 CipherSpecs require that the the personal certificate must have an RSA public key. When these CipherSpecs are used, RSA key exchange is used to establish the secret key for the connection.

This list of restrictions is not exhaustive: depending on the configuration, there might be additional restrictions which can further affect the ability to interoperate. For example, if IBM MQ is configured to comply with the FIPS 140-2 or NSA Suite B standards then this will also limit the range of allowable configurations. Refer to the following section for more information.

If you need to use different types of CipherSpec on the same queue manager or client application, configure an appropriate certificate label and CipherSpec combination on the client definition.

The three types of CipherSpec do not interoperate directly: this is a limitation of the current SSL and TLS standards. For example, suppose you have chosen to use the ECDHE_ECDSA_AES_128_CBC_SHA256 CipherSpec for a receiver channel named TO.QM1 on a queue manager named QM1.

ECDHE_ECDSA_AES_128_CBC_SHA256 is a Type 1 CipherSpec, so the remote sender end of channel TO.QM1 must have a personal certificate with an Elliptic Curve key and an ECDSA-based digital signature. If the remote sender channel does not meet these requirements the channel fails to start.

Other channels connecting to queue manager QM1 can use other CipherSpecs, provided that each channel uses a certificate of the correct type for the CipherSpec of that channel. For example, suppose that QM1 uses a sender channel named TO.QM2 to send messages to another queue manager named QM2. The channel TO.QM2 could use the Type 3 CipherSpec TLS_RSA_WITH_AES_256_CBC_SHA256 provided both ends of the channel use certificates containing RSA public keys. The certificate label channel attribute can be used to configure a different certificate for each channel.

When planning your IBM MQ networks, consider carefully which channels require SSL or TLS, and ensure that the type of certificates used for each channel are appropriate for use with the CipherSpec on that channel.

To view the digital signature algorithm and public key type for a digital certificate you can use the **runmqakm** command:

```
runmqakm -cert -details -db key.kdb -pw password -label cert_label
```

where *cert_label* is the label of the certificate whose digital signature algorithm you need to display. See Digital certificate labels for details.

The execution of the **runmqakm** command will produce output displaying the Public Key Type:

```
Label : ibmwebspheremqexample
Key Size : 384
Version : X509 V3
Serial : 9ad5eeef5d756f41
Issuer : CN=Example Certificate Authority,OU=Test,O=Example,C=US
Subject : CN=Example Queue Manager,OU=Test,O=Example,C=US
Not Before : 21 August 2011 13:10:24 GMT+01:00
Not After : 21 August 2012 13:10:24 GMT+01:00
Public Key
    30 76 30 10 06 07 2A 86 48 CE 3D 02 01 06 05 2B
    81 04 00 22 03 62 00 04 3E 6F A9 06 B6 C3 A0 11
    F8 D6 22 78 FE EF 0A FE 34 52 C0 8E AB 5E 81 73
    D0 97 3B AB D6 80 08 E7 31 E9 18 3F 6B DE 06 A7
    15 D6 9D 5B 6F 56 3B 7F 72 BB 6F 1E C9 45 1C 46
    60 BE F2 DC 1B AD AC EC 64 4C 0E 06 65 6E ED 93
    B8 F5 95 E0 F9 2A 05 D6 21 02 BD FB 06 63 A1 CC
    66 C6 8A 0A 5C 3F F7 D3
Public Key Type : EC_ecPublicKey (1.2.840.10045.2.1)
Fingerprint : SHA1 :
    3C 34 58 04 5B 63 5F 5C C9 7A E7 67 08 2B 84 43
    3D 43 7A 79
Fingerprint : MD5 :
    49 13 13 E1 B2 AC 18 9A 31 41 DC 8C B4 D6 06 68
Fingerprint : SHA256 :
    6F 76 78 68 F3 70 F1 53 CE 39 31 D9 05 C5 C5 9F
    F2 B8 EE 21 49 16 1D 90 64 6D AC EB 0C A7 74 17
Signature Algorithm : EC_ecdsa_with_SHA384 (1.2.840.10045.4.3.3)
```

```
Value
    30 65 02 30 0A B0 2F 72 39 9E 24 5A 22 FE AC 95
    0D 0C 6D 6C 2F B3 E7 81 F6 C1 36 1B 9A B0 6F 07
    59 2A A1 4C 02 13 7E DD 06 D6 FE 4B E4 03 BC B1
    AC 49 54 1E 02 31 00 90 0E 46 2B 04 37 EE 2C 5F
    1B 9C 69 E5 99 60 84 84 10 71 1A DA 63 88 33 E2
    22 CC E6 1A 4E F4 61 CC 51 F9 EE A0 8E F4 DC B5
    0B B9 72 58 C3 C7 A4
Trust Status : Enabled
```

The Public Key Type line in this case shows that the certificate has an Elliptic Curve public key. The Signature Algorithm line in this case shows that the EC_ecdsa_with_SHA384 algorithm is in use: this is based upon the ECDSA algorithm. This certificate is therefore only suitable for use with Type 1 CipherSpecs.

You can also use the **iKeycmd (runmqckm)** tool with the same parameters. Also the **iKeyman (strmqikm)** GUI can be used to view digital signature algorithms if you open the key repository and double-click the label of the certificate. However, you are advised to use the **runmqakm** tool to view digital certificates because it supports a wider range of algorithms.

Elliptic Curve CipherSpecs and NSA Suite B

When IBM MQ is configured to conform to the Suite B compliant TLS 1.2 profile, the permitted CipherSpecs and digital signature algorithms are restricted as described in “NSA Suite B Cryptography in IBM MQ” on page 415. Additionally, the range of acceptable Elliptic Curve keys is reduced according to the configured security levels.

At the 128-bit Suite B security level, the certificate subject's public key is required to use either the NIST P-256 or NIST P-384 elliptic curve and to be signed with either the NIST P-256 elliptic curve or the NIST P-384 elliptic curve. The **runmqakm** command can be used to request digital certificates for this security level using a -sig_alg parameter of EC_ecdsa_with_SHA256, or EC_ecdsa_with_SHA384.

At the 192-bit Suite B security level, the certificate subject's public key is required to use the NIST P-384 elliptic curve and to be signed with the NIST P-384 elliptic curve. The **runmqakm** command can be used to request digital certificates for this security level using a -sig_alg parameter of EC_ecdsa_with_SHA384.

The supported NIST elliptic curves are as follows:

Table 19. Supported NIST elliptic curves

NIST FIPS 186-3 curve name	RFC 4492 curve name	Elliptic Curve key size (bits)
P-256	secp256r1	256
P-384	secp384r1	384
P-521	secp521r1	521

Note: The NIST P-521 elliptic curve cannot be used for Suite B compliant operation.

Related concepts:

"Enabling CipherSpecs" on page 797
Enable a CipherSpec by using the **SSLCIPH** parameter in either the **DEFINE CHANNEL** MQSC command or the **ALTER CHANNEL** MQSC command.

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 653
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

"NSA Suite B Cryptography in IBM MQ" on page 415
This topic provides information about how to configure IBM MQ on Windows, Linux, and UNIX systems to conform to the Suite B compliant TLS 1.2 profile.

"National Security Agency (NSA) Suite B Cryptography" on page 394
The government of the Unites States of America produces technical advice on IT systems and security, including data encryption. The US National Security Agency (NSA) recommends a set of interoperable cryptographic algorithms in its Suite B standard.

Channel authentication records

To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records.

You might find that clients attempt to connect to your queue manager using a blank user ID or a high-level user ID that would allow the client to perform undesirable actions. You can block access to these clients using channel authentication records. Alternatively, a client might assert a user ID that is valid on the client platform but is unknown or of an invalid format on the server platform. You can use a channel authentication record to map the asserted user ID to a valid user ID.

You might find a client application that connects to your queue manager and behaves badly in some way. To protect the server from the issues this application is causing, it needs to be temporarily blocked using the IP address the client application is on until such time as the firewall rules are updated or the client application is corrected. You can use a channel authentication record to block the IP address from which the client application connects.

If you have set up an administration tool such as the MQ Explorer, and a channel for that specific use, you might want to ensure that only specific client computers can use it. You can use a channel authentication record to allow the channel to be used only from certain IP addresses.

If you are just getting started with some sample applications running as clients, see Preparing and running the sample programs for an example of setting up the queue manager securely using channel authentication records.

To get channel authentication records to control inbound channels, use the MQSC command **ALTER QMGR CHLAUTH(ENABLED)**.

Channel authentication records can be created to perform the following functions:

- To block connections from specific IP addresses.
- To block connections from specific user IDs.
- To set an MCAUSER value to be used for any channel connecting from a specific IP address.
- To set an MCAUSER value to be used for any channel asserting a specific user ID.
- To set an MCAUSER value to be used for any channel having a specific SSL or TLS Distinguished Name (DN).
- To set an MCAUSER value to be used for any channel connecting from a specific queue manager.
- To block connections claiming to be from a certain queue manager unless the connection is from a specific IP address.

- To block connections presenting a certain SSL or TLS certificate unless the connection is from a specific IP address.

These uses are explained further in the following sections.

You create, modify, or remove channel authentication records using the MQSC command **`SET CHLAUTH`** or the PCF command **`Set Channel Authentication Record`**.

Note: Large numbers of channel authentication records can have a negative impact on a queue manager's performance.

Blocking IP addresses

It is normally the role of a firewall to prevent access from certain IP addresses. However, there might be occasions where you experience connection attempts from an IP address that should not have access to your IBM MQ system and must temporarily block the address before the firewall can be updated. These connection attempts might not be coming from IBM MQ channels; these connection attempts might be coming from other socket applications that are misconfigured to target your IBM MQ listener. Block IP addresses by setting a channel authentication record of type BLOCKADDR. You can specify one or more single addresses, ranges of addresses, or patterns including wildcards.

Whenever an inbound connection is refused because the IP address is blocked in this manner, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_ADDRESS is issued, provided that channel events are enabled and the queue manager is running. Additionally, the connection is held open for 30 seconds prior to returning the error to ensure the listener is not flooded with repeated attempts to connect that are blocked.

To block IP addresses only on specific channels, or to avoid the delay before the error is reported, set a channel authentication record of type ADDRESSMAP with the USERSRC(NOACCESS) parameter.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See “Blocking specific IP addresses” on page 762 for an example.

Blocking user IDs

To prevent certain user IDs from connecting over a client channel, set a channel authentication record of type BLOCKUSER. This type of channel authentication record applies only to client channels, not to message channels. You can specify one or more individual user IDs to be blocked, but you cannot use wildcards.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_USERID is issued, provided that channel events are enabled.

See “Blocking specific user IDs” on page 763 for an example.

You can also block any access for specified user IDs on certain channels by setting a channel authentication record of type USERMAP with the USERSRC(NOACCESS) parameter.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See “Blocking access for a client asserted user ID” on page 766 for an example.

Blocking queue manager names

To specify that any channel connecting from a specified queue manager is to have no access, set a channel authentication record of type QMGRMAP with the USERSRC(NOACCESS) parameter. You can specify a single queue manager name or a pattern including wildcards. There is no equivalent of the BLOCKUSER function to block access from queue managers.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See “Blocking access from a remote queue manager” on page 766 for an example.

Blocking SSL or TLS DNs

To specify that any user presenting an SSL or TLS personal certificate containing a specified DN is to have no access, set a channel authentication record of type SSLPEERMAP with the USERSRC(NOACCESS) parameter. You can specify a single distinguished name or a pattern including wildcards. There is no equivalent of the BLOCKUSER function to block access for DNs.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See “Blocking access for an SSL Distinguished Name” on page 767 for an example.

Mapping IP addresses to user IDs to be used

To specify that any channel connecting from a specified IP address is to use a specific MCAUSER, set a channel authentication record of type ADDRESSMAP. You can specify a single address, a range of addresses, or a pattern including wildcards.

If you use a port forwarder, DMZ session break, or any other setup which changes the IP address presented to the queue manager, then mapping IP addresses is not necessarily suitable for your use.

See “Mapping an IP address to an MCAUSER user ID” on page 767 for an example.

Mapping queue manager names to user IDs to be used

To specify that any channel connecting from a specified queue manager is to use a specific MCAUSER, set a channel authentication record of type QMGRMAP. You can specify a single queue manager name or a pattern including wildcards.

See “Mapping a remote queue manager to an MCAUSER user ID” on page 764 for an example.

Mapping user IDs asserted by a client to user IDs to be used

To specify that if a certain user ID is used by a connection from an IBM MQ MQI client, a different, specified MCAUSER is to be used, set a channel authentication record of type USERMAP. User ID mapping does not use wildcards.

See “Mapping a client asserted user ID to an MCAUSER user ID” on page 765 for an example.

Mapping SSL or TLS DNs to user IDs to be used

To specify that any user presenting an SSL/TLS personal certificate containing a specified DN is to use a specific MCAUSER, set a channel authentication record of type SSLPEERMAP. You can specify a single distinguished name or a pattern including wildcards.

See "Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID" on page 765 for an example.

Mapping queue managers, clients, or SSL or TLS DNs according to IP address

In some circumstances it might be possible for a third party to spoof a queue manager name. An SSL or TLS certificate or key database file might also be stolen and reused. To protect against these threats, you can specify that a connection from a certain queue manager or client, or using a certain DN must be connecting from a specified IP address. Set a channel authentication record of type USERMAP, QMGRMAP or SSLPEERMAP and specify the permitted IP address, or pattern of IP addresses, using the ADDRESS parameter.

See "Mapping a remote queue manager to an MCAUSER user ID" on page 764 for an example.

Interaction between channel authentication records

It is possible that a channel attempting to make a connection matches more than one channel authentication record, and that these have contradictory effects. For example, a channel might assert a user ID which is blocked by a BLOCKUSER channel authentication record, but with an SSL or TLS certificate that matches an SSLPEERMAP record that sets a different user ID. In addition, if channel authentication records use wildcards, a single IP address, queue manager name, or SSL or TLS DN might match several patterns. For example, the IP address 192.0.2.6 matches the patterns 192.0.2.0-24, 192.0.2.*, and 192.0.*.6. The action taken is determined as follows.

- The channel authentication record used is selected as follows:
  - A channel authentication record explicitly matching the channel name takes priority over a channel authentication record matching the channel name by using a wildcard.
  - A channel authentication record using an SSL or TLS DN takes priority over a record using a user ID, queue manager name, or IP address.
  - A channel authentication record using a user ID or queue manager name takes priority over a record using an IP address.
- If a matching channel authentication record is found and it specifies an MCAUSER, this MCAUSER is assigned to the channel.
- If a matching channel authentication record is found and it specifies that the channel has no access, an MCAUSER value of *NOACCESS is assigned to the channel. This value can later be changed by a security exit program.
- If no matching channel authentication record is found, or a matching channel authentication record is found and it specifies that the user ID of the channel is to be used, the MCAUSER field is inspected.
  - If the MCAUSER field is blank, the client user ID is assigned to the channel.
  - If the MCAUSER field is not blank, it is assigned to the channel.
- Any security exit program is run. This exit program might set the channel user ID or determine that access is to be blocked.
- If the connection is blocked or the MCAUSER is set to *NOACCESS, the channel ends.
- If the connection is not blocked, for any channel except a client channel, the channel user ID determined in the previous steps is checked against the list of blocked users.
  - If the user ID is in the list of blocked users, the channel ends.
  - If the user ID is not in the list of blocked users, the channel runs.

Where a number of channel authentication records match a channel name, IP address, hostname, queue manager name, or SSL or TLS DN, the most specific match is used. The match considered to be:

- The most specific is a name without wildcard characters, for example:
  - A channel name of A.B.C
  - An IP address of 192.0.2.6
  - A hostname of `hursley.ibm.com`
  - A queue manager name of 192.0.2.6
- The most generic is a single asterisk (*) that matches, for example:
  - All channel names
  - All IP addresses
  - All hostnames
  - All queue manager names
- A pattern with an asterisk at the start of a string is more generic than a defined value at the start of a string:
  - For channels, *.B.C is more generic than A.*
  - For IP addresses, *.0.2.6 is more generic than 192.*
  - For hostnames, `*.ibm.com` is more generic than `hursley.*`
  - For queue manager names, *QUEUEMANAGER is more generic than QUEUEMANAGER*
- A pattern with an asterisk at a specific place in a string is more generic than a defined value at the same place in a string, and similarly for each subsequent place in a string:
  - For channels, A.*.C is more generic than A.B.*
  - For IP addresses, 192.*.2.6 is more generic than 192.0.*.
  - For hostnames, `hursley.*.com` is more generic than `hursley.ibm.*`
  - For queue manager names, Q*MANAGER is more generic than QUEUE*
- Where two or more patterns have an asterisk at a specific place in a string, the one with fewer nodes following the asterisk is more generic:
  - For channels, A.* is more generic than A.*.C
  - For IP addresses, 192.* is more generic than 192.*.2.*.
  - For hostnames, `hurlsey.*` is more generic than `hursley.*.com`
  - For queue manager names, Q* is more generic than Q*MGR
- Additionally, for an IP address:
  - A range indicated with a hyphen (-), is more specific than an asterisk. Thus 192.0.2.0-24 is more specific than 192.0.2.*.
  - A range that is a subset of another is more specific than the larger range. Thus 192.0.2.5-15 is more specific than 192.0.2.0-24.
  - Overlapping ranges are not permitted. For example, you cannot have channel authentication records for both 192.0.2.0-15 and 192.0.2.10-20.
  - A pattern cannot have fewer than the required number of parts, unless the pattern ends with a single trailing asterisk. For example 192.0.2 is invalid, but 192.0.2.* is valid.
  - A trailing asterisk must be separated from the rest of the address by the appropriate part separator (a dot (.) for IPv4, a colon (:) for IPv6). For example, 192.0* is not valid because the asterisk is not in a part of its own.
  - A pattern can contain additional asterisks, provided that no asterisk is adjacent to the trailing asterisk. For example, 192.*.2.* is valid, but 192.0.*.* is not valid.
  - An IPv6 address pattern cannot contain a double colon and a trailing asterisk, because the resulting address would be ambiguous. For example, 2001::* could expand to 2001:0000:*, 2001:0000:0000:* and so on

- For an SSL or TLS Distinguished Name (DN), the precedence order of substrings is as follows:

Table 20. Precedence order of substrings

Order	DN substring	Name
1	SERIALNUMBER=	Certificate serial number
2	MAIL=	Email address
3	E=	Email address (Deprecated in preference to MAIL)
4	UID=, USERID=	User identifier
5	CN=	Common name
6	T=	Title
7	OU=	Organizational unit
8	DC=	Domain component
9	O=	Organization
10	STREET=	Street / First line of address
11	L=	Locality
12	ST=, SP=, S=	State or province name
13	PC=	Postal code / zip code
14	C=	Country
15	UNSTRUCTUREDNAME=	Host name
16	UNSTRUCTUREDADDRESS=	IP address
17	DNQ=	Distinguished name qualifier

Thus, if an SSL or TLS certificate is presented with a DN containing the substrings O=IBM and C=UK, IBM MQ uses a channel authentication record for O=IBM in preference to one for C=UK, if both are present.

A DN can contain multiple OUs, which must be specified in hierarchical order with the large organizational units specified first. If two DNs are equal in all respects except for their OU values, the more specific DN is determined as follows:

1. If they have different numbers of OU attributes then the DN with the most OU values is more specific. This is because the DN with more Organizational Units fully qualifies the DN in more detail and provides more matching criteria. Even if its top-level OU is a wildcard (OU=*), the DN with more OUs is still regarded as more specific overall.
2. If they have the same number of OU attributes then the corresponding pairs of OU values are compared in sequence left-to-right, where the left-most OU is the highest-level (least specific), according to the following rules.
   a. An OU with no wildcard values is the most specific because it can only match exactly one string.
   b. An OU with a single wildcard at either the beginning or end (for example, OU=ABC* or OU=*ABC) is next most specific.
   c. An OU with two wildcards for example, OU=*ABC*) is next most specific.
   d. An OU consisting only of an asterisk (OU=*) is the least specific.
3. If the string comparison is tied between two attribute values of the same specificity then whichever attribute string is longer is more specific.
4. If the string comparison is tied between two attribute values of the same specificity and length then the result is determined by a case-insensitive string comparison of the portion of the DN excluding any wildcards.

If two DNs are equal in all respects except for their DC values, the same matching rules apply as for OUs except that in DC values the left-most DC is the lowest-level (most specific) and the comparison ordering differs accordingly.

Displaying channel authentication records

To display channel authentication records, use the MQSC command **`DISPLAY CHLAUTH`** or the PCF command **`Inquire Channel Authentication Records`**. You can choose to return all records that match the supplied channel name, or you can choose an explicit match. The explicit match tells you which channel authentication record would be used if a channel attempted to make a connection from a specific IP address, from a specific queue manager or using a specific user ID, and, optionally, presenting an SSL/TLS personal certificate containing a specified DN.

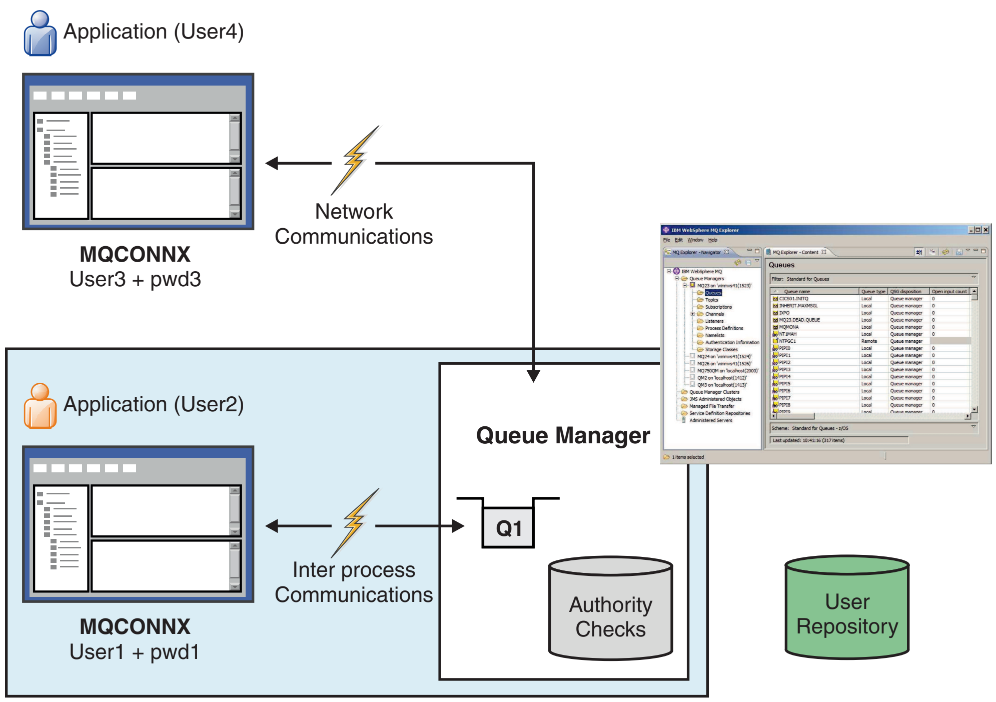
Related concepts:

"Security for remote messaging" on page 463
This section deals with remote messaging aspects of security.

Connection authentication

Connection authentication can be achieved in various ways:

- An application can provide a user ID and password. The application can be either a client, or it can use local bindings.
- A queue manager can be configured to act on a supplied user ID and password.
- A repository can be used to determine whether a user ID and password combination is valid.



In the diagram, two applications are making connections with a queue manager, one application as a client and one using local bindings. Applications might use a variety of APIs to connect to the queue manager, but all have the ability to provide a user ID and a password. The user ID that the application is running under, `User1` and `user3` in the diagram, which is the usual operating system user ID presented to IBM MQ, might be different from the user ID provided by the application, `User2` and `user4`.

The queue manager receives configuration commands (in the diagram, MQ Explorer is being used) and manages the opening of resources and checks the authority to access those resources. There are many different resources in IBM MQ that an application might require authority to access. The diagram illustrates opening a queue for output, but the same principles apply to other resources as well.

See User repositories for details about the repository that is used for checking user IDs and passwords.

Related concepts:

"Connection authentication: Configuration"
A queue manager can be configured to use a supplied user ID and password to check whether a user has authority to access resources.

"Connection authentication: Application changes" on page 435

"Connection authentication: User repositories" on page 436
For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Connection authentication: Configuration:

A queue manager can be configured to use a supplied user ID and password to check whether a user has authority to access resources.

Turning on connection authentication on a queue manager

On a queue manager object, the **CONNAUTH** attribute can be set to the name of an authentication information (AUTHINFO) object. This object can be one of two types (AUTHTYPE attribute):

IDPWOS
Indicates that the queue manager uses the local operating system to authenticate the user ID and password.

IDPWLDAP
Indicates that the queue manager uses an LDAP server to authenticate the user ID and password.

Note: You cannot use any other type of authentication information object in the **CONNAUTH** field.

IDPWOS and IDPWLDAP are similar in a number of their attributes, which are described here. Other attributes are considered later.

To check local connections, use the AUTHINFO attribute **CHCKLOCL** (check local connections). To check client connections, use the AUTHINFO attribute **CHCKCLNT** (check client connections). The configuration must be refreshed before the queue manager recognizes the changes.

```
ALTER QMGR CONNAUTH(USE.PW)
DEFINE AUTHINFO(USE.PW) +
AUTHTYPE(IDPWOS) +
FAILDLAY(10) +
CHCKLOCL(OPTIONAL) +
CHCKCLNT(REQUIRED)
REFRESH SECURITY TYPE(CONNAUTH)
```

Where `USE.PW` in the CONNAUTH is a string that matches the AUTHINFO definition.

Both **CHCKLOCL** and **CHCKCLNT** have the same set of possible values that allow the strictness of checking to be varied:

NONE Switches off checking.

OPTIONAL
Ensures that if a user ID and password are provided by an application, they are a valid pair, but that it is not mandatory to provide them. This option might be useful during migration, for example.

Important: `OPTIONAL` is the minimum value you can set, in order to use more stringent CHLAUTH rules.

If you select `NONE` and the client connection matches a CHLAUTH record with CHCKCLNT `REQUIRED` (or `REQDADM` on platforms other than z/OS), the connection fails. You receive message AMQ9793 on platforms other than z/OS, and message CSQX793E on z/OS.

REQUIRED
Requires that all applications provide a valid user ID and password. See also the following note.

REQDADM
Privileged users must supply a valid user ID and password, but non-privileged users are treated as with the `OPTIONAL` setting. See also the following note. z/OS (This setting is not allowed on z/OS systems.)

Note:

Setting **CHCKLOCL** to `REQUIRED` or `REQDADM` means that you cannot locally administer the queue manager by using **runmqsc** (error AMQ8135: Not authorized) unless the user specifies the `-u UserId` parameter on the **runmqsc** command line. With that set, **runmqsc** prompts for the user's password at the console.

Similarly, a user running IBM MQ Explorer on the local system will see error AMQ4036 when attempting to connect to the queue manager. To specify a user name and password, right-click the local queue manager object and select **Connection Details** > **Properties...** from the menu. In the **Userid** section, enter the user name and password to be used, then click **OK**.

Similar considerations apply to remote connections with **CHCKCLNT**.

CONNAUTH is blank for migrated queue managers but set to *SYSTEM.DEFAULT.AUTHINFO.IDPWOS* for new queue managers. The preceding **AUTHINFO** definition has **CHCKCLNT** set to *REQDADM* by default.

Therefore, you need to provide the correct operating system password for any existing clients using a privileged user ID to connect.

Note: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see "MQCSP password protection" on page 404.

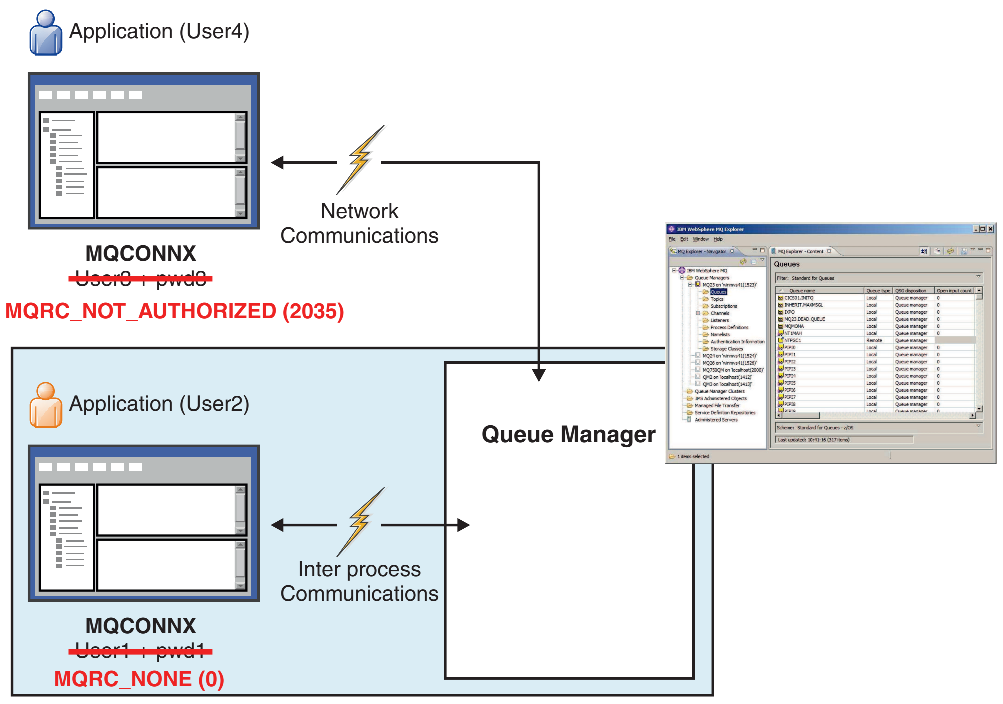
Configuration granularity

In addition to **CHCKLOCL** and **CHCKCLNT** that are used to turn on user ID and password checking, there are enhancements to the CHLAUTH rules so that more specific configuration can be made using **CHCKCLNT**.

You can set the overall **CHCKCLNT** value to `OPTIONAL`, for example, and then upgrade it to be more stringent for certain channels by setting **CHCKCLNT** to `REQUIRED` or `REQDADM` on the CHLAUTH rule. By default, CHLAUTH rules will run with CHCKCLNT(ASQMGR) so this granularity does not have to be used. For example:

```
DEFINE AUTHINFO(USE.PW) AUTHTYPE(xxxxxx) +
CHCKCLNT(OPTIONAL)
SET CHLAUTH('*') TYPE(ADDRESSMAP) +
ADDRESS('*') USERSRC(CHANNEL) +
CHCKCLNT(REQUIRED)
SET CHLAUTH('*') TYPE(SSLPEERMAP) +
SSLPEER('CN=*') USERSRC(CHANNEL)
```

Error notification



An error is recorded if an application does not supply a user ID and password when required to, or supplies an incorrect combination even when it is optional.

Note: When password checking is turned off, by using the `NONE` option on either **`CHCKLOCL`** or **`CHCKCLNT`**, invalid passwords are not detected.

Failed authentications are held for the number of seconds specified by the **`FAILDLAY`** attribute before the error is returned to the application. This provides some protection from an application repeatedly trying to connect.

The error is recorded in a number of ways:

Application
The application is returned the standard IBM MQ security error, RC2035 - MQRC_NOT_AUTHORIZED.

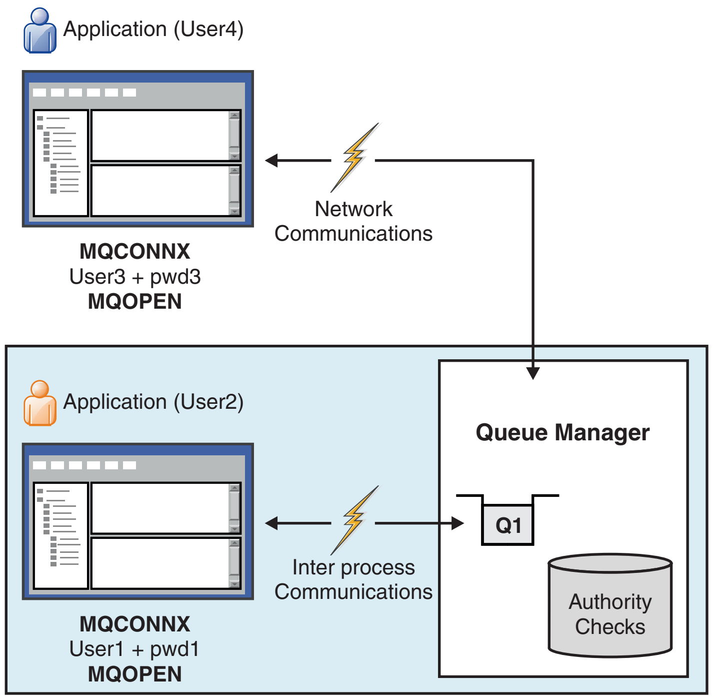
Administrator
An IBM MQ administrator sees the event reported in the error log and can therefore see that the application was rejected because the user ID and password failed the check, rather than because, for example, there was no connection authority .

Monitoring tool
A monitoring tool can also be notified of the failure, if you turn on authority events by sending an event message to the SYSTEM.ADMIN.QMGR.EVENT queue:

```
ALTER QMGR AUTHOREV(ENABLED)
```

This "Not Authorized" event is a Type 1 connect event, and provides the same fields as other Type 1 events, with an additional field, the MQCSP user ID that was provided. The password is not given in the event message. This means that there are two user IDs in the event message: the ID that the application is running under and the ID that the application presented for user ID and password checking.

Relationship to authorization



You can configure a queue manager to mandate that user IDs and passwords are provided by certain applications as the user ID that the application is running under might not be the same user ID that was presented by the application along with a password when the application opens a queue for output, for example:

```
ALTER QMGR CONNAUTH(USE.PWD)
DEFINE AUTHINFO(USE.PWD) +
AUTHTYPE(xxxxxx) +
CHCKLOCL(OPTIONAL) +
CHCKCLNT(REQUIRED) +
ADOPTCTX(YES)
```

How user IDs and passwords are handled is controlled by the **ADOPTCTX** attribute on the authentication information object.

ADOPTCTX(YES)
All authorization checks for an application are made with the same user ID that you authenticated by password, by selecting to adopt the context as the application context for the rest of the life of the connection.

ADOPTCTX(NO)
An application provides a user ID and password for the purposes of authenticating them at connection time, but then continues by using the user ID that the application is running under for future authorization checks. You might find this option useful when migrating. You might also find it useful to continue using this option, perhaps with client connections, because authorization checks are done by using an assigned message channel agent user identifier (MCAUSER) based on an IP address.

The **ADOPTCTX** setting has no effect if the user ID presented for authentication by password is the same user ID that the application is also running under.

Attention: V 8.0.0.5

When you use the **ADOPTCTX(YES)** parameter on an authentication information object, another security context cannot be adopted unless you set the **ChlauthEarlyAdopt** parameter in the channels stanza of the `qm.ini` file.

For example, the default authentication information object is set to **ADOPTCTX(YES)**, and the user `fred` is logged in. The following two CHLAUTH rules are configured:

```
SET CHLAUTH('MY.CHLAUTH') TYPE(ADDRESSMAP) DESCR('Block all access by
default') ADDRESS('*') USERSRC(NOACCESS) ACTION(REPLACE)
SET CHLAUTH('MY.CHLAUTH') TYPE(USERMAP) DESCR('Allow user bob and force
CONNAUTH') CLNTUSER('bob') CHCKCLNT(REQUIRED) USERSRC(CHANNEL)
```

The following command is issued, with the intention of authenticating the command as the adopted security context of the user `bob`:

```
runmqsc -c -u bob QMGR
```

In fact, the queue manager uses the security context of `fred`, not `bob`, and the connection fails.

For more information about **ChlauthEarlyAdopt**, see Attributes of the channels stanza.

Related concepts:

"Connection authentication" on page 430

"Connection authentication: Application changes"

"Connection authentication: User repositories" on page 436
For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Connection authentication: Application changes:

An application can provide a user ID and password within the connection security parameters (MQCSP) structure when MQCONNX is called. The user ID and password are passed for checking to the object authority manager (OAM) supplied with the queue manager, or the authorization service component supplied with the queue manager on z/OS systems. You do not have to write your own custom interface.

If the application is running as a client, the user ID and password are also passed to the client-side and server-side security exits for processing. They can also be used for setting the message channel agent user identifier (MCAUSER) attribute of a channel instance. The security exit is called with exit reason MQXR_SEC_PARMS for this processing. Client-side security exits and the pre-connect exit, can make changes to MQCONN before it is sent to the queue manager.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see "MQCSP password protection" on page 404.

By using the XAOPEN string to provide a user ID and password, you can avoid having to make changes to the application code.

Note:

From IBM WebSphere MQ Version 6.0, the security exit has allowed the MQCSP to be set. Therefore, clients at this level or later do not have to be upgraded.

However, in versions of IBM MQ prior to Version 8.0, MQCSP placed no restrictions on the user ID and password that were provided by the application. When using these values with features provided by IBM MQ there are limits which apply to the use of these features, but if you are only passing them to your own exits, those limits do not apply.

Related concepts:

"Connection authentication" on page 430

"Connection authentication: Configuration" on page 431
A queue manager can be configured to use a supplied user ID and password to check whether a user has authority to access resources.

"Connection authentication: User repositories"
For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Connection authentication: User repositories:

For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

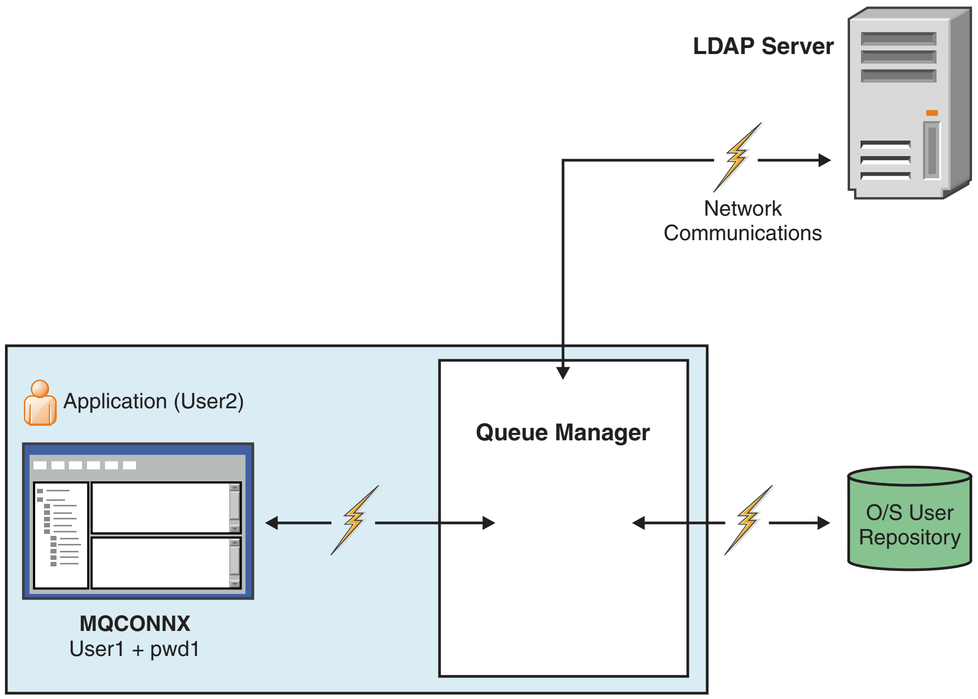


Figure 60. Types of authentication information objects

```
DEFINE AUTHINFO(USE.OS) AUTHTYPE(IDPWOS)
DEFINE AUTHINFO(USE.LDAP) +
AUTHTYPE(IDPWLDAP) +
CONNAME('ldap1(389),ldap2(389)') +
LDAPUSER('CN=QMGR1') +
LDAPPWD('passw0rd') SECCOMM(YES)
```

There are two types of authentication information object, as represented in the diagram:

- IDPWOS is used to indicate that the queue manager uses the local operating system to authentication the user ID and password. If you choose to use the local operating system, you only need to set the common attributes, as described in the preceding topics.
- IDPWLDAP is used to indicate that the queue manager uses an LDAP server to authenticate the user ID and password. If you choose to use an LDAP server, more information is provided below.

Only one type of authentication information object can be chosen for each queue manager to use, by naming the appropriate object in the queue manager's **CONNAUTH** attribute.

Using an LDAP server for authentication.

Set the **CONNAME** field to the address of the LDAP server for the queue manager. You can provide additional addresses for the LDAP server in a comma-separated list, which can help with redundancy if the LDAP server does not provide this facility itself.

Set the required LDAP server ID and password in the **LDAPUSER** and **LDAPPWD** fields so that the queue manager can access the LDAP server and look up information about user records.

Secure connection to an LDAP Server

Unlike channels, there is no **SSLCIPH** parameter to turn on the use of SSL/TLS for communication with the LDAP server. In this case IBM MQ is acting as a client to the LDAP server so much of the configuration is done at the LDAP server. Some existing parameters in IBM MQ are used to configure how that connection works.

Set the **SECCOMM** field to control whether connectivity to the LDAP server uses SSL/TLS.

In addition to this attribute, the queue manager attributes **SSLFIPS** and **SUITEB** restrict the set of cipher specs that are chosen. The certificate that is used to identify the queue manager to the LDAP server is the queue manager certificate, either `ibmwebspheremq` *<qmgr-name>* or the value of the **CERTLABL** attribute. See Digital certificate labels for details.

LDAP User Repository

When using an LDAP user repository there is some more configuration to be done on the queue manager other than just to tell the queue manager where to find the LDAP server.

User IDs defined in an LDAP server have a hierarchical structure that uniquely identifies them. An application can therefore connect to the queue manager and present its user ID as the fully qualified hierarchical user ID.

However, to simplify the information that an application has to provide, it is possible to configure the queue manager to assume that the first part of the hierarchy is common to all IDs, and to automatically add this before the shortened ID provide by the application. The queue manager can then present a complete ID to the LDAP server.

Set BASEDNU to the initial point that the LDAP search looks for the ID in the LDAP hierarchy. When you set BASEDNU, you must ensure that only one result is returned when you search for the ID in the LDAP hierarchy.

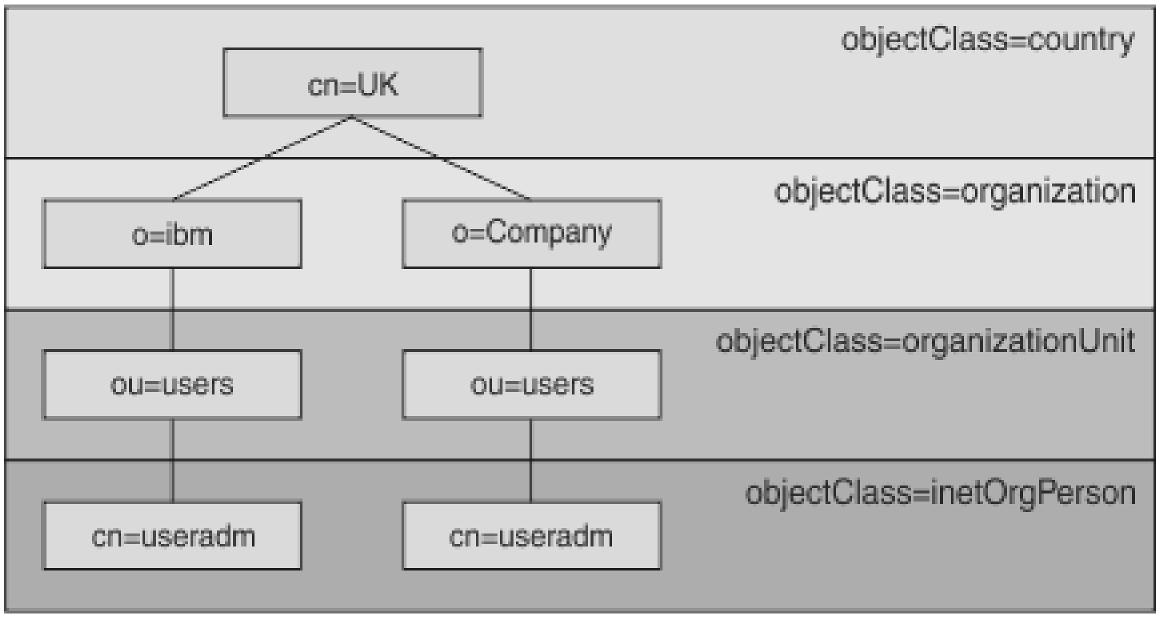


Figure 61. An example LDAP hierarchy

For example, in Figure 61 BASEDNU can be set to "ou=users,o=ibm,c=UK" or ",o=ibm,c=UK". However, because a distinguished name that contains "cn=useradm" exists in both the "o=ibm" branch and the "o=Company" branch, BASEDNU cannot be set to "c=UK". For performance and security reasons, use the highest point in your LDAP hierarchy that from which you can reference all of the userids that you need. In this example, that is "ou=users,o=ibm,c=UK".

Your application might submit to the queue manager the user ID without providing the LDAP attribute name, `CN=` for example. If you set USRFIELD to the LDAP attribute name, this value is added as a prefix to the user ID that comes from the application. This might be a useful migratory aid when moving from operating system user IDs to LDAP user IDs, as the application can then present the same string in both cases and you can avoid making changes to the application.

The complete user ID presented to the LDAP server therefore looks like this:

```
<USRFIELD> = <ID_from_application> <BASEDNU>
```

Related concepts:

"Connection authentication" on page 430

"Connection authentication: Configuration" on page 431
A queue manager can be configured to use a supplied user ID and password to check whether a user has authority to access resources.

"Connection authentication: Application changes" on page 435

Client side security exit to insert user ID and password (`mqccred`):

If you have any client applications that are required to send a user ID or password but you are unable to change the source yet, there is a security exit shipped with IBM MQ Version 8.0 called **`mqccred`** that you can use. **`mqccred`** provides a user ID and password on behalf of the client application, from a `.ini` file. This user ID and password are sent to the queue manager which, if configured to do so, will authenticate them.

Overview

`mqccred` is a security exit that runs on the same machine as your client application. It allows user ID and password information to be supplied on behalf of the client application, where that information is not being supplied by the application itself. The user ID and password information is supplied in a structure known as the Connection Security Parameters (MQCSP) and will be authenticated by the queue manager if connection authentication is configured.

User ID and password information is retrieved from a `.ini` file on the client machine. The passwords in the file are protected by obfuscation using the **`runmqccred`** command, and also by ensuring the file permissions on the `.ini` file are set such that only the user ID running the client application (and therefore the exit) are able to read it.

Location

`mqccred` is installed:

Windows platforms
In the `<installation directory>\Tools\c\Samples\mqccred\` directory

UNIX platforms
In the `<installation_directory>/samp/mqccred` directory

Notes: The exit:

1. Acts purely as a security channel exit, and needs to be the only such exit defined on a channel.
2. Is usually named through the Client Channel Definition Table (CCDT), but a Java client can have the exit mentioned in the JNDI objects directly, or the exit might be configured for applications that manually construct the MQCD structure.
3. You must copy the **`mqccred`** and **`mqccred_r`** programs to the `var/mqm/exits` directory.

   For example, on a 64 bit UNIX platform machine, issue the command:

   ```
   cp <installation_directory>/samp/mqccred/lib64/* /var/mqm/exits
   ```

   See A step by step example of how to test mqccred for more information.
4. Is capable of running on previous versions of IBM MQ ; as far back as Version 7.0.1.

Setting up user IDs and passwords

The `.ini` file contains stanzas for each queue manager, with a global setting for unspecified queue managers. Each stanza contains the name of the queue manager, a user ID, and either a plain text or obfuscated password.

You must edit the `.ini` file by hand, using whichever editor you want, and add the plain text password attribute to the stanzas. Run the provided, **runmqccred** program, which takes the `.ini` file and replaces the **Password** attribute with the **OPW** attribute, an obfuscated form of the password.

See runmqccred for a description of the command and its parameters.

The `mqccred.ini` file contains your user ID and password information.

A template `.ini` file is provided in the same directory as the exit to provide a starting point for your enterprise.

By default, this file will be looked for in `$HOME/.mqs/mqccred.ini`. If you would like to locate it elsewhere, you can use the environment variable *MQCCRED* to point at it:

```
MQCCRED=C:\mydir\mqccred.ini
```

If you use *MQCCRED*, the variable must include the full name of the configuration file, including any `.ini` filetype. Since this file contains passwords (even if obfuscated), you are expected to protect the file using operating system privileges to ensure unauthorized people cannot read it. If you do not have the correct file permission, the exit will not run successfully.

If the application has already supplied an MQCSP structure, the exit normally respects this and will not insert any information from the `.ini` file. However, you can override this by using the **Force** attribute in the stanza.

Setting **Force** to the value *TRUE* removes the application-supplied user ID and password, and replaces those with the ini file version.

You can also set the **Force** attribute in the global section of the file to set the default value of that file.

The default value for **Force** is *FALSE*.

You can provide a user ID and password for all queue managers, or for each individual queue manager. This is an example of an `mqccred.ini` file:

```
# comments are permitted
AllQueueManagers:
User=abc
OPW=%^&aervrgtsr

QueueManager:
Name=QMA
User=user1
OPW=H&^dbgfh

Force=TRUE

QueueManager:
Name=QMB
User=user2
password=passw0rd
```

Notes:

1. The individual queue manager definitions take precedence over the global setting.

2. Attributes are case insensitive.

Constraints

When this exit is in use, the local user ID of the person running the application does not flow from the client to the server. The only identity information available is from the ini file contents.

Therefore, you must configure the queue manager to either use **ADOPTCTX(YES)**, or map the inbound connection request to an appropriate user ID through one of the available mechanisms, for example, "Channel authentication records" on page 424.

Important: If you add new passwords, or update old ones, the **runmqccred** command only processes any plain text passwords, leaving your obfuscated ones untouched.

Debugging

The exit writes to the standard IBM MQ trace when that is enabled.

To assist in debugging configuration issues, the exit can also write directly to stdout.

No channel security exit data (**SCYDATA**) configuration is normally required for the channel. However, you can specify:

ERROR
: Only print information abut error conditions, such as not being able to find the configuration file.

DEBUG
: Displays these error conditions, and some additional trace statements.

NOCHECKS
: Bypasses the constraints on file permissions, and the further constraint that the `.ini` file should not contain any unprotected passwords.

You can put one or more of these elements into the **SCYDATA** field, separated by commas, in any order. For example, `SCYDATA=(NOCHECKS,DEBUG)`.

Note that the items are case-sensitive, and must be entered in uppercase.

Using `mqccred`

Once you have your file set up, you can invoke the channel exit by updating your client-connection channel definition to include the `SCYEXIT('mqccred(ChlExit)')` attribute:

```
DEFINE CHANNEL(<channelname>) CHLTYPE(clntconn) +
CONNAME(<remote machine>) +
QMNAME(<remote qmgr>) +
SCYEXIT('mqccred(ChlExit)') +
REPLACE
```

Related information:
SCYDATA
SCYEXIT
runmqccred

Connection authentication with the Java client:

Connection authentication is a feature in IBM MQ that allows the queue manager to be configured, to authenticate applications, using a provided user ID and password. When the application is a Java application using client bindings, there are two modes in which this can be run.

Compatibility mode

In releases prior to IBM MQ Version 8.0, the Java client could send a user ID and password across the client-connection channel to the server-connection channel, and have them provided to a security exit in the **`RemoteUserIdentifier`** and **`RemotePassword`** fields of the MQCD structure. In compatibility mode, this behavior is retained.

You might use this mode in combination with connection authentication, and migrate away from any security exits that were previously used to do the same job.

You must use ADOPTCTX(YES) or have another method, for example a CHLAUTH rule based on an SSL/TLS certificate, to set the running MCAUSER when using compatibility mode, as in this mode, the client side user ID is not sent to the queue manager.

Compatibility mode is the default setting.

MQCSP authentication mode

In this mode, the client side user ID is sent as well as the user ID and password to be authenticated, so you are able to use ADOPTCTX(NO). The user ID and password are available to a server-connection security exit in the MQCSP structure provided in the MQCXP structure.

This mode of operation can be enabled on a connection-by-connection basis or globally:
- In IBM MQ classes for Java, set the property *MQConstants.USE_MQCSP_AUTHENTICATION_PROPERTY* to **true** in the properties hashtable passed to the **`com.ibm.mq.MQQueueManager`** constructor.
- In IBM MQ classes for JMS, set the property *JmsConstants.USER_AUTHENTICATION_MQCSP* to **true**, on the appropriate connection factory prior to creating the connection.
- Globally, set the system property *com.ibm.mq.cfg.jmqi.useMQCSPauthentication* to a value indicating true, for example, by adding `-Dcom.ibm.mq.cfg.jmqi.useMQCSPauthentication=Y` to the command line.

Enabling MQCSP Authentication mode in IBM MQ Explorer

The MQ Explorer is a Java application, so these two modes are applicable to it as well.

As for other Java applications, compatibility mode is the default.

On panels where user identification is provided, there is a check box to enable or disable compatibility mode.

By default, this check box is enabled. To use MQCSP Authentication, remove the check mark from this box.

Related concepts:

"Connection authentication" on page 430

"Connection authentication: Application changes" on page 435

"Connection authentication: User repositories" on page 436
For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Message security in IBM MQ

Message security in IBM MQ infrastructure is provided by a separately licensed component IBM MQ Advanced Message Security.

IBM MQ Advanced Message Security (IBM MQ AMS) expands IBM MQ security services to provide data signing and encryption at the message level. The expanded services guarantees that message data has not been modified between when it is originally placed on a queue and when it is retrieved. In addition, IBM MQ AMS verifies that a sender of message data is authorized to place signed messages on a target queue.

Related concepts:

"IBM MQ Advanced Message Security" on page 849
IBM MQ Advanced Message Security (IBM MQ AMS) is a separately licensed component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

Planning for your security requirements

This collection of topics explains what you need to consider when planning security in an IBM MQ environment.

You can use IBM MQ for a wide variety of applications on a range of platforms. The security requirements are likely to be different for each application. For some, security will be a critical consideration.

IBM MQ provides a range of link-level security services, including support for the Secure Sockets Layer (SSL) and Transport Layer Security (TLS).

You must consider certain aspects of security when implementing WebSphere. On IBM i IBM i, UNIX, Linux and Windows systems, if you ignore these aspects and do nothing, you cannot use IBM MQ. z/OS On z/OS, the effect of ignoring these aspects is that your IBM MQ resources are unprotected. That is, all users can access and change all IBM MQ resources.

Authority to administer IBM MQ

IBM MQ administrators need authority to:

- Issue commands to administer IBM MQ
- Use the IBM MQ Explorer
- z/OS Use the operations and control panels on z/OS
- IBM i Use IBM i administrative panels and commands.
- z/OS Use the IBM MQ utility program, CSQUTIL, on z/OS
- z/OS Access the queue manager data sets on z/OS

For more information, see:

- "Authority to administer IBM MQ on UNIX, Linux, and Windows systems" on page 777
- IBM i "Authority to administer IBM MQ on IBM i" on page 449
- z/OS "Authority to administer IBM MQ on z/OS" on page 450

Authority to work with IBM MQ objects

Applications can access the following IBM MQ objects by issuing MQI calls:

- Queue managers
- Queues
- Processes
- Namelists
- Topics

Applications can also use Programmable Command Format (PCF) commands to access these IBM MQ objects, and to access channels and authentication information objects as well. These objects can be protected by IBM MQ so that the user IDs associated with the applications need authority to access them.

For more information, see "Authorization for applications to use IBM MQ" on page 452.

Channel security

The user IDs associated with message channel agents (MCAs) need authority to access various IBM MQ resources. For example, an MCA must be able to connect to a queue manager. If it is a sending MCA, it must be able to open the transmission queue for the channel. If it is a receiving MCA, it must be able to open destination queues. The user IDs associated with applications which need to administer channels, channel initiators, and listeners need authority to use the relevant PCF commands. However, most applications do not need such access.

For more information, see "Channel authorization" on page 485.

Additional considerations

You need to consider the following aspects of security only if you are using certain IBM MQ function or base product extensions:

- "Security for queue manager clusters" on page 499
- "Security for IBM MQ Publish/Subscribe" on page 501
- "Security for IBM MQ internet pass-thru" on page 505

Planning identification and authentication

Decide what user IDs to use, and how and at what levels you want to apply authentication controls.

You must decide how you will identify the users of your IBM MQ applications, bearing in mind that different operating systems support user IDs of different lengths. You can use channel authentication records to map from one user ID to another, or to specify a user ID based on some attribute of the connection. IBM MQ channels using SSL or TLS use digital certificates as a mechanism for identification and authentication. Each digital certificate has a subject distinguished name which can be mapped onto specific identities using channel authentication records. Additionally, CA certificates in the key repository determine which digital certificates may be used to authenticate to IBM MQ. For more information see:

- "Mapping a remote queue manager to an MCAUSER user ID" on page 764
- "Mapping a client asserted user ID to an MCAUSER user ID" on page 765
- "Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID" on page 765
- "Mapping an IP address to an MCAUSER user ID" on page 767

Planning authentication for a client application

You can apply authentication controls at four levels: at the communications level, in security exits, with channel authentication records, and in terms of the identification that is passed to a security exit.

There are four levels of security to consider. The diagram shows an IBM MQ MQI client that is connected to a server. Security is applied at four levels, as described in the following text. MCA is a Message Channel Agent.

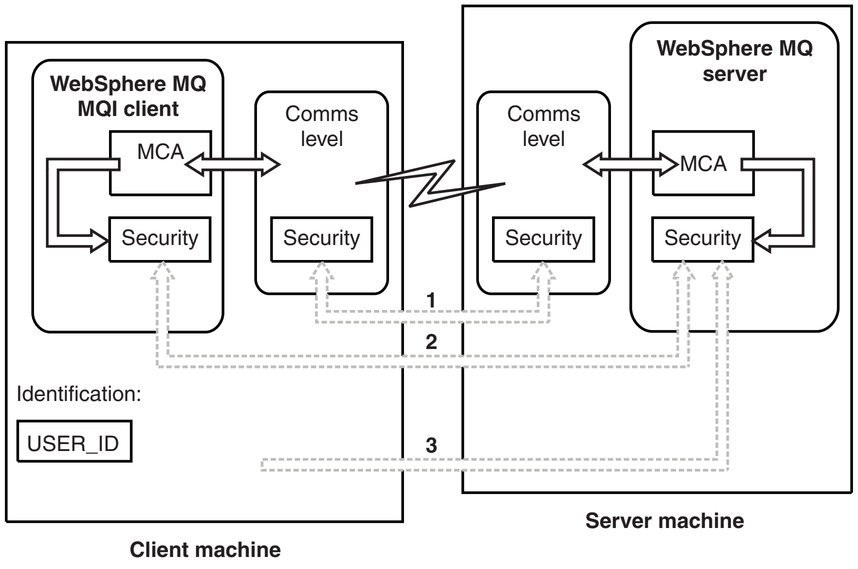


Figure 62. Security in a client/server connection

1. Communications level

   See arrow 1. To implement security at the communications level, use SSL or TLS. For more information, see "Cryptographic security protocols: SSL and TLS" on page 388
2. Channel authentication records

   See arrows 2 & 3. Authentication can be controlled by using the IP address or SSL/TLS distinguished names at the security level. A user ID can also be blocked or an asserted user ID can be mapped to a valid user ID. A full description is given in "Channel authentication records" on page 424.
3. Connection authentication

   See arrow 3. The client sends an ID and a password. For more information, see “Connection authentication: Configuration” on page 431.
4. Channel security exits

   See arrow 2. The channel security exits for client to server communication can work in the same way as for server to server communication. A protocol independent pair of exits can be written to provide mutual authentication of both the client and the server. A full description is given in Channel security exit programs.
5. Identification that is passed to a channel security exit

   See arrow 3. In client to server communication, the channel security exits do not have to operate as a pair. The exit on the IBM MQ client side can be omitted. In this case, the user ID is placed in the channel descriptor (MQCD) and the server-side security exit can alter it, if required.

   Windows clients also send extra information to assist identification.
   - The user ID that is passed to the server is the currently logged-on user ID on the client.
   - The security ID of the currently logged-on user.

   To assist identification on IBM MQ client for HP Integrity NonStop Server, the client passes the OSS Safeguard alias under which the client application is running. This ID is typically of the form `<PRIMARYGROUP>.<ALIAS>`. If required, you can map this user ID to an alternative user ID on the queue manager by using either channel authentication records or a security exit. For more information about message exits, see “Identity mapping in message exits” on page 720. For more information about defining channel authentication records, see “Mapping a client asserted user ID to an MCAUSER user ID” on page 765.

   The values of the user ID and, if available, the security ID, can be used by the server security exit to establish the identity of the IBM MQ MQI client.

From IBM MQ Version 8.0, you can send passwords that are included in the MQCSP structure.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see “MQCSP password protection” on page 404.

User IDs:

When you create user IDs for client applications, the user IDs must not be longer than the maximum permitted length. You must not use the reserved user IDs UNKNOWN and NOBODY. If the server that the client connects to is an IBM MQ for Windows server, you must escape the use of the at sign, @. The permitted length of user IDs is dependent on the platform that is used for the server:
- On z/OS and UNIX and Linux, the maximum length of a user ID is 12 characters.
- IBM i On IBM i, the maximum length of a user ID is 10 characters.
- On Windows, if both the IBM MQ MQI client, and the IBM MQ server are on Windows, and the server has access to the domain on which the client user ID is defined, the maximum length of a user ID is 20 characters. However, if the IBM MQ server is not a Windows server, the user ID is truncated to 12 characters. You can use the MQCSP structure to avoid the truncation of the user ID.
- If you use the MQCSP structure to pass credentials, the maximum length of a user ID is 1024 characters. For more information about the MQCSP structure, see “Identifying and authenticating users using the MQCSP structure” on page 718.

Although user IDs are used to authenticate, groups are used for authorization, except for Windows.

If you create service accounts, without paying attention to groups, and authorize all the user IDs differently, every user can access the information of every other user.

Restricted user IDs

The user IDs `UNKNOWN` and `NOBODY` have special meanings to IBM MQ. Creating user IDs in the operating system called `UNKNOWN` or `NOBODY` could have unintended results.

User IDs when connecting to an IBM MQ for Windows server

An IBM MQ for Windows server does not support the connection of a Windows client if the client is running under a user ID that contains the @ character, for example, `abc@d`. The return code to the **MQCONN** call at the client is MQRC_NOT_AUTHORIZED.

However, you can specify the user ID using two @ characters, for example, `abc@@d`. Using the `id@domain` format is the preferred practice, to ensure that the user ID is resolved in the correct domain consistently; thus `abc@@d@domain`.

Planning authorization

Plan the users who will have administrative authority and plan how to authorize users of applications to appropriately use IBM MQ objects, including those connecting from an IBM MQ MQI client.

Individuals or applications must be granted access in order to use IBM MQ. What access they require depend on the roles they undertake and the tasks which they need to perform. Authorization in IBM MQ can be subdivided into two main categories:

- Authorization to perform administrative operations
- Authorization for applications to use IBM MQ

Both classes of operation are controlled by the same component and an individual can be granted authority to perform both categories of operation.

The following topics give further information about specific areas of authorization that you must consider:

Authority to administer IBM MQ

IBM MQ administrators need authority to perform various functions. This authority is obtained in different ways on different platforms.

IBM MQ administrators need authority to:

- Issue commands to administer IBM MQ
- Use the IBM MQ Explorer
- z/OS Use the operations and control panels on z/OS
- z/OS Use the IBM MQ utility program, CSQUTIL, on z/OS
- z/OS Access the queue manager data sets on z/OS

For more information, see the topic appropriate to your operating system.

Authority to administer IBM MQ on UNIX and Windows systems:

An IBM MQ administrator is a member of the mqm group. This group has access to all IBM MQ resources and can issue IBM MQ control commands. An administrator can grant specific authorities to other users.

To be an IBM MQ administrator on UNIX and Windows systems, a user must be a member of the *mqm group*. This group is created automatically when you install IBM MQ. To allow users to issue control commands, you must add them to the mqm group. This includes the root user on UNIX systems.

Users who are not member of the mqm group can be granted administrative privileges, but they are not able to issue IBM MQ control commands, and they are authorized to execute only the commands for which they have been granted access.

Additionally, on Windows systems, the SYSTEM and Administrator accounts have full access to IBM MQ resources.

All members of the mqm group have access to all IBM MQ resources on the system, including being able to administer any queue manager running on the system. This access can be revoked only by removing a user from the mqm group. On Windows systems, members of the Administrators group also have access to all IBM MQ resources.

Administrators can use the control command **runmqsc** to issue IBM MQ Script (MQSC) commands. When **runmqsc** is used in indirect mode to send MQSC commands to a remote queue manager, each MQSC command is encapsulated within an Escape PCF command. Administrators must have the required authorities for the MQSC commands to be processed by the remote queue manager.

The IBM MQ Explorer issues PCF commands to perform administration tasks. Administrators require no additional authorities to use the IBM MQ Explorer to administer a queue manager on the local system. When the IBM MQ Explorer is used to administer a queue manager on another system, administrators must have the required authorities for the PCF commands to be processed by the remote queue manager.

For more information about the authority checks carried out when PCF and MQSC commands are processed, see the following topics:
- For commands that operate on queue managers, queues, channels, processes, namelists, and authentication information objects, see "Authorization for applications to use IBM MQ" on page 452.
- For commands that operate on channels, channel initiators, listeners, and clusters, see Channel security.
- z/OS For MQSC commands that are processed by the command server on IBM MQ for z/OS, see "Command security and command resource security" on page 450.

For more information about the authority you need to administer IBM MQ on UNIX and Windows systems, see the related information.

Authority to administer IBM MQ on IBM i:

To be an IBM MQ administrator on IBM i, you must be a member of the *QMQMADM group*. This group has properties similar to those of the mqm group on UNIX and Windows systems. In particular, the QMQMADM group is created when you install IBM MQ for IBM i, and members of the QMQMADM group have access to all IBM MQ resources on the system. You also have access to all IBM MQ resources if you have *ALLOBJ authority.

Administrators can use CL commands to administer IBM MQ. One of these commands is GRTMQMAUT, which is used to grant authorities to other users. Another command, STRMQMMQSC, enables an administrator to issue MQSC commands to a local queue manager.

There are two groups of CL command provided by IBM MQ for IBM i:

Group 1
: To issue a command in this category, a user must be a member of the QMQMADM group or have *ALLOBJ authority. GRTMQMAUT and STRMQMMQSC belong to this category, for example.

Group 2
: To issue a command in this category, a user does not need to be a member of the QMQMADM group or have *ALLOBJ authority. Instead, two levels of authority are required:

  - The user requires IBM i authority to use the command. This authority is granted by using the GRTOBJAUT command.
  - The user requires IBM MQ authority to access any IBM MQ object associated with the command. This authority is granted by using the GRTMQMAUT command.

  The following are examples of commands in this group:

  - CRTMQMQ, Create MQM Queue
  - CHGMQMPRC, Change MQM Process
  - DLTMQMNL, Delete MQM Namelist
  - DSPMQMAUTI, Display MQM Authentication Information
  - CRTMQMCHL, Create MQM channel

  For more information about this group of commands, see “Authorization for applications to use IBM MQ” on page 452.

For a complete list of group 1 and group 2 commands, see “Access authorities for IBM MQ objects on IBM i” on page 540

For more information about the authority you need to administer IBM MQ on IBM i, see Administering IBM i .

Authority to administer IBM MQ on z/OS:

This collection of topics describes various aspects of the authority you need to administer IBM MQ for z/OS.

Authority checks on z/OS:

IBM MQ uses the System Authorization Facility (SAF) to route requests for authority checks to an external security manager (ESM) such as the z/OS Security Server Resource Access Control Facility (RACF). IBM MQ does no authority checks of its own.

It is assumed that you are using RACF as your ESM. If you are using a different ESM, you might need to interpret the information provided for RACF in a way that is relevant to your ESM.

You can specify whether you want authority checks turned on or off for each queue manager individually or for every queue manager in a queue-sharing group. This level of control is called *subsystem security*. If you turn subsystem security off for a particular queue manager, no authority checks are carried out for that queue manager.

If you turn subsystem security on for a particular queue manager, authority checks can be performed at two levels:

Queue-sharing group level security
Authority checks use RACF profiles that are shared by all queue managers in the queue-sharing group. This means that there are fewer profiles to define and maintain, making security administration easier.

Queue manager level security
Authority checks use RACF profiles specific to the queue manager.

You can use a combination of queue-sharing group and queue manager level security. For example, you can arrange for profiles specific to a queue manager to override those of the queue-sharing group to which it belongs.

Subsystem security, queue-sharing group level security, and queue manager level security are turned on or off by defining *switch profiles*. A switch profile is a normal RACF profile that has a special meaning to IBM MQ.

Command security and command resource security:

Command security relates to the authority to issue a command; command resource authority relates to the authority to perform an operation on a resource. Both are implemented y using RACF classes.

Authority checks are carried out when an IBM MQ administrator issues an MQSC command. This is called *command security*.

To implement command security, you must define certain RACF profiles and give the necessary groups and user IDs access to these profiles at the required levels. The name of a profile for command security contains the name of an MQSC command.

Some MQSC commands perform an operation on an IBM MQ resource, such as the DEFINE QLOCAL command to create a local queue. When an administrator issues an MQSC command, authority checks are carried out to determine whether the requested operation can be performed on the resource specified in the command. This is called *command resource security*.

To implement command resource security, you must define certain RACF profiles and give the necessary groups and user IDs access to these profiles at the required levels. The name of a profile for command resource security contains the name of an IBM MQ resource and its type (QUEUE, PROCESS, NAMELIST, TOPIC, AUTHINFO, or CHANNEL).

Command security and command resource security are independent. For example, when an administrator issues the command:

```
DEFINE QLOCAL(MOON.EUROPA)
```

the following authority checks are performed:

- Command security checks that the administrator is authorized to issue the DEFINE QLOCAL command.
- Command resource security checks that the administrator is authorized to perform an operation on the local queue called MOON.EUROPA.

Command security and command resource security can be turned on or off by defining switch profiles.

MQSC commands and the system command input queue:

Use this topic to understand how the command server processes MQSC commands directed to the system command input queue.

Command security and command resource security are also used when the command server retrieves a message containing an MQSC command from the system command input queue. The user ID that is used for the authority checks is the one found in the *UserIdentifier* field in the message descriptor of the message containing the MQSC command. This user ID must have the required authorities on the queue manager where the command is processed. For more information about the *UserIdentifier* field and how it is set, see Message context.

Messages containing MQSC commands are sent to the system command input queue in the following circumstances:

- The operations and control panels send MQSC commands to the system command input queue of the target queue manager. The MQSC commands correspond to the actions you choose on the panels. The *UserIdentifier* field in each message is set to the TSO user ID of the administrator.
- The COMMAND function of the IBM MQ utility program, CSQUTIL, sends the MQSC commands in the input data set to the system command input queue of the target queue manager. The COPY and EMPTY functions send DISPLAY QUEUE and DISPLAY STGCLASS commands. The *UserIdentifier* field in each message is set to the job user ID.
- The MQSC commands in the CSQINPX data sets are sent to the system command input queue of the queue manager to which the channel initiator is connected. The *UserIdentifier* field in each message is set to the channel initiator address space user ID.

  No authority checks are performed when MQSC commands are issued from the CSQINP1 and CSQINP2 data sets. You can control who is allowed to update these data sets using RACF data set protection.
- Within a queue-sharing group, a channel initiator might send START CHANNEL commands to the system command input queue of the queue manager to which it is connected. A command is sent when an outbound channel that uses a shared transmission queue is started by triggering. The *UserIdentifier* field in each message is set to the channel initiator address space user ID.
- An application can send MQSC commands to a system command input queue. By default, the *UserIdentifier* field in each message is set to the user ID associated with the application.
- On UNIX, Linux and Windows systems, the **runmqsc** control command can be used in indirect mode to send MQSC commands to the system command input queue of a queue manager on z/OS. The *UserIdentifier* field in each message is set to the user ID of the administrator who issued the **runmqsc** command.

Access to the queue manager data sets:

IBM MQ administrators need authority to access the queue manager data sets. Use this topic to understand which data sets need RACF protection.

These data sets include:
- The data sets referred to by CSQINP1, CSQINP2, CSQINPT, and CSQXLIB in the queue manager's started task procedure
- The queue manager's page sets, active log data sets, archive log data sets, and bootstrap data sets (BSDSs)
- The data sets referred to by CSQXLIB and CSQINPX in the channel initiator's started task procedure

You must protect the data sets so that no unauthorized user can start a queue manager or gain access to any queue manager data. To do this, use RACF data set protection.

Authorization for applications to use IBM MQ

When applications access objects, the user IDs associated with the applications need appropriate authority.

Applications can access the following IBM MQ objects by issuing MQI calls:
- Queue managers
- Queues
- Processes
- Namelists
- Topics

Applications can also use PCF commands to administer IBM MQ objects. When the PCF command is processed, it uses the authority context of the user ID that put the PCF message.

Applications, in this context, include those written by users and vendors, and those supplied with IBM MQ for z/OS. The applications supplied with IBM MQ for z/OS include:
- The operations and control panels
- The IBM MQ utility program, CSQUTIL
- The dead letter queue handler utility, CSQUDLQH

Applications that use IBM MQ classes for Java, IBM MQ classes for JMS, IBM MQ classes for .NET, or the Message Service Clients for C/C++ and .NET use the MQI indirectly.

MCAs also issue MQI calls and the user IDs associated with the MCAs need authority to access these IBM MQ objects. For more information about these user IDs and the authorities they require, see "Channel authorization" on page 485.

On z/OS, applications can also use MQSC commands to access these IBM MQ objects but command security and command resource security provide the authority checks in these circumstances.
z/OS For more information, see "Command security and command resource security" on page 450 and "MQSC commands and the system command input queue" on page 451.

On IBM i, a user that issues a CL command in Group 2 might require authority to access an IBM MQ object associated with the command. For more information, see "When authority checks are performed" on page 453.

When authority checks are performed:

Authority checks are performed when an application attempts to access a queue manager, queue, process, or namelist.

On IBM i, authority checks might also be performed when a user issues a CL command in Group 2 that accesses any of these IBM MQ objects. The checks are performed in the following circumstances:

When an application connects to a queue manager using an MQCONN or MQCONNX call

The queue manager asks the operating system for the user ID associated with the application. The queue manager then checks that the user ID is authorized to connect to it and retains the user ID for future checks.

Users do not have to sign on to IBM MQ. IBM MQ assumes that users are signed on to the underlying operating system and are been authenticated by it.

When an application opens an IBM MQ object using an MQOPEN or MQPUT1 call

All authority checks are performed when an object is opened, not when it is accessed later. For example, authority checks are performed when an application opens a queue. They are not performed when the application puts messages on the queue or gets messages from the queue.

When an application opens an object, it specifies the types of operation it needs to perform on the object. For example, an application might open a queue to browse the messages on it, get messages from it, but not to put messages on it. For each type of operation, the queue manager checks that the user ID associated with the application has the authority to perform that operation.

When an application opens a queue, the authority checks are performed against the object named in the `ObjectName` field of the object descriptor. The `ObjectName` field is used on the MQOPEN or MQPUT1 calls. If the object is an alias queue or a remote queue definition, the authority checks are performed against the object itself. They are not performed on the queue to which the alias queue or the remote queue definition resolves. This means that the user does not need permission to access it. Limit the authority to create queues to privileged users. If you do not, users might bypass the normal access control simply by creating an alias.

An application can reference a remote queue explicitly. It sets the `ObjectName` and `ObjectQMgrName` fields in the object descriptor to the names of the remote queue and the remote queue manager. The authority checks are performed against the transmission queue with the same name as the remote queue manager. On z/OS, a check is made on the RACF queue profile that matches the remote queue manager name. On platforms other than z/OS, a check is made against the RQMNAME profile that matches the remote queue manager name, if clustering is being used. An application can reference a cluster queue explicitly by setting the `ObjectName` field in the object descriptor to the name of the cluster queue. The authority checks are performed against the cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

The authority to a dynamic queue is based on the model queue from which it is derived, but is not necessarily the same; see note 1.

The user ID that the queue manager uses for the authority checks is obtained from the operating system. The user ID is obtained when the application connects to the queue manager. A suitably authorized application can issue an MQOPEN call specifying an alternative user ID; access control checks are then made on the alternative user ID. Using an alternate user ID does not change the user ID associated with the application, only the one used for access control checks.

When an application subscribes to a topic using an MQSUB call

When an application subscribes to a topic, it specifies the type of operation that it needs to perform. It is either creating a subscription, altering an existing subscription, or resuming an existing subscription without changing it. For each type of operation, the queue manager checks that the user ID that is associated with the application has the authority to perform the operation.

When an application subscribes to a topic, the authority checks are performed against topic objects that are found in the topic tree. The topic objects are at, or above, the point in the topic tree at which the application subscribed. The authority checks might involve checks on more than one topic object. The user ID that the queue manager uses for the authority checks is obtained from the operating system. The user ID is obtained when the application connects to the queue manager.

The queue manager performs authority checks on subscriber queues but not on managed queues.

When an application deletes a permanent dynamic queue using an MQCLOSE call

The object handle specified on the MQCLOSE call is not necessarily the same one returned by the MQOPEN call that created the permanent dynamic queue. If it is different, the queue manager checks the user ID associated with the application that issued the MQCLOSE call. It checks that the user ID is authorized to delete the queue.

When an application that closes a subscription to remove it did not create it, the appropriate authority is required to remove it.

When a PCF command that operates on an IBM MQ object is processed by the command server

This rule includes the case where a PCF command operates on an authentication information object.

The user ID that is used for the authority checks is the one found in the `UserIdentifier` field in the message descriptor of the PCF command. This user ID must have the required authorities on the queue manager where the command is processed. The equivalent MQSC command encapsulated within an Escape PCF command is treated in the same way. For more information about the `UserIdentifier` field, and how it is set, see "Message context" on page 455.

IBM i **On IBM i, when a user issues a CL command in Group 2 that operates on an IBM MQ object**

This rule includes the case where a CL command in Group 2 operates on an authentication information object.

Checks are performed to determine whether the user has the authority to operate on an IBM MQ object associated with the command. The checks are performed unless the user is a member of the QMQMADM group or has `*ALLOBJ` authority. The authority required depends on the type of operation that the command performs on the object. For example, the command **`CHGMQMQ`**, Change MQM Queue, requires the authority to change the attributes of the queue specified by the command. In contrast, the command **`DSPMQMQ`**, Display MQM Queue, requires the authority to display the attributes of the queue specified by the command.

Many commands operate on more than one object. For example, to issue the command **`DLTMQMQ`**, Delete MQM Queue, the following authorities are required:

- The authority to connect to the queue manager specified by the command
- The authority to delete the queue specified by the command

Some commands operate on no object at all. In this case, the user requires only IBM i authority to issue one of these commands. **`STRMQMLSR`**, Start MQM Listener, is an example of such a command.

Alternate user authority:

When an application opens an object or subscribes to a topic, the application can supply a user ID on the MQOPEN, MQPUT1, or MQSUB call. It can ask the queue manager to use this user ID for authority checks instead of the one associated with the application.

The application succeeds in opening the object only if both the following conditions are met:
- The user ID associated with the application has the authority to supply a different user ID for authority checks. The application is said to have *alternate user authority*.
- The user ID supplied by the application has the authority to open the object for the types of operation requested, or to subscribe to the topic.

Message context:

Message context information allows the application that retrieves a message to find out about the originator of the message. The information is held in fields in the message descriptor and the fields are divided into three logical parts

These parts are as follows:

identity context
These fields contain information about the user of the application that put the message on the queue.

origin context
These fields contain information about the application itself and when the message was put on the queue.

user context
These fields contain message properties that applications can use to select messages that the queue manager should deliver.

When an application puts a message on a queue, the application can ask the queue manager to generate the context information in the message. This is the default action. Alternatively, it can specify that the context fields are to contain no information. The user ID associated with an application requires no special authority to do either of these.

An application can set the identity context fields in a message, allowing the queue manager to generate the origin context, or it can set all the context fields. An application can also pass the identity context fields from a message it has retrieved to a message it is putting on a queue, or it can pass all the context fields. However, the user ID associated with an application requires authority to set or pass context information. An application specifies that it intends to set or pass context information when it opens the queue on which it is about to put messages, and its authority is checked at this time.

Here is a brief description of each of the context fields:

Identity context

UserIdentifier
The user ID associated with the application that put the message. If the queue manager sets this field, it is set to the user ID obtained from the operating system when the application connects to the queue manager.

AccountingToken
Information that can be used to charge for the work done as a result of the message.

ApplIdentityData
If the user ID associated with an application has authority to set the identity context

fields, or to set all the context fields, the application can set this field to any value related to identity. If the queue manager sets this field, it is set to blank.

Origin context

PutApplType
The type of the application that put the message; a CICS transaction, for example.

PutApplName
The name of the application that put the message.

PutDate
The date when the message was put.

PutTime
The time when the message was put.

ApplOriginData
If the user ID associated with an application has authority to set all the context fields, the application can set this field to any value related to origin. If the queue manager sets this field, it is set to blank.

User context

The following values are supported for **MQINQMP** or **MQSETMP**:

MQPD_USER _CONTEXT

The property is associated with the user context.

No special authorization is required to be able to set a property associated with the user context using the MQSETMP call.

On a V7.0 or subsequent queue manager, a property associated with the user context is saved as described for MQOO_SAVE_ALL_CONTEXT. An MQPUT with MQOO_PASS_ALL_CONTEXT specified causes the property to be copied from the saved context into the new message.

MQPD_NO_CONTEXT

The property is not associated with a message context.

An unrecognized value is rejected with MQRC_PD_ERROR. The initial value of this field is **MQPD_NO_CONTEXT**.

For a detailed description of each of the context fields, see MQMD - Message descriptor. For more information about how to use message context, see Message context.

Authority to work with IBM MQ objects on : IBM i IBM i ,UNIXLinuxWindows

The authorization service component provided with IBM MQ is called the *object authority manager (OAM)*. It provides access control via authentication and authorization checks.

- Authentication.

  The authentication check performed by the OAM provided with IBM MQ is basic, and is only performed in specific circumstances. It is not intended to meet the strict requirements expected in a highly secure environment.

  The OAM performs its authentication check when an application connects to a queue manager, and the following conditions are true.

  If an MQCSP structure has been supplied by the connecting application, and the *AuthenticationType* attribute in the MQCSP structure is given the value MQCSP_AUTH_USER_ID_AND_PWD, then the check is performed by the OAM in its MQZID_AUTHENTICATE_USER function. This is the check: the user ID in the MQCSP structure is compared against the user ID in the `IdentityContext` (MQZIC), to determine whether they match. If they do not match, the check fails.

  This basic check is not intended to be a full authentication of the user. For example, there is no check of the authenticity of the user by checking the password supplied in the MQCSP structure. Also, if the application omits an MQCSP structure, then no check is performed.

  If fuller authentication services are required in the queue manager via the authorization service component, then the OAM provided with IBM MQ does not offer this. You must write a new authorization service component, or obtain one from a vendor.
- Authorization.

  The authorization checks are comprehensive, and are intended to meet most normal requirements.

  Authorization checks are performed when an application issues an MQI call to access a queue manager, queue, process, topic, or namelist. They are also performed at other times, for example, when a command is being performed by the Command Server.

On IBM i IBM i , UNIX, Linux and Windows systems, the *authorization service* provides the access control when an application issues an MQI call to access an IBM MQ object that is a queue manager, queue, process, topic, or namelist. This includes checks for alternative user authority and the authority to set or pass context information.

On Windows , the OAM gives members of the Administrators group the authority to access all IBM MQ objects, even when UAC is enabled.

Additionally, on Windows systems, the SYSTEM account has full access to IBM MQ resources.

The authorization service also provides authority checks when a PCF command operates on one of these IBM MQ objects or an authentication information object. The equivalent MQSC command encapsulated within an Escape PCF command is treated in the same way.

IBM i On IBM i , unless the user is a member of the QMQMADM group or has *ALLOBJ authority, the authorization service also provides authority checks when a user issues a CL command in Group 2 that operates on any of these IBM MQ objects or an authentication information object.

The authorization service is an *installable service*, which means that it is implemented by one or more *installable service components*. Each component is invoked using a documented interface. This enables users and vendors to provide components to augment or replace those provided by the IBM MQ products.

The authorization service component provided with IBM MQ is called the *object authority manager (OAM)*. The OAM is automatically enabled for each queue manager you create.

The OAM maintains an access control list (ACL) for each IBM MQ object it is controlling access to. On UNIX and Linux systems, only group IDs can appear in an ACL. This means that all members of a group have the same authorities. On IBM i IBM i and on Windows systems, both user IDs and group IDs can appear in an ACL. This means that authorities can be granted to individual users and groups.

A 12 character limitation applies to both the group and the user ID. UNIX platforms generally restrict the length of a user ID to 12 characters. AIX and Linux have raised this limit but IBM MQ continues to observe a 12 character restriction on all UNIX platforms. If you use a user ID of greater than 12 characters, IBM MQ replaces it with the value "UNKNOWN". Do not define a user ID with a value of "UNKNOWN".

The OAM can authenticate a user and change appropriate identity context fields. You enable this by specifying a connection security parameters structure (MQCSP) on an MQCONNX call. The structure is passed to the OAM Authenticate User function (MQZ_AUTHENTICATE_USER), which sets appropriate identity context fields. If an MQCONNX connection from an IBM MQ client, the information in the MQCSP is flowed to the queue manager to which the client is connecting over the client-connection and server-connection channel. If security exits are defined on that channel, the MQCSP is passed into each security exit and can be altered by the exit. Security exits can also create the MQCSP. For more details of the use of security exits in this context, see Channel security exit programs.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see IBM MQCSP password protection.

On UNIX, Linux and Windows systems, the control command **setmqaut** grants and revokes authorities and is used to maintain the ACLs. For example, the command:

```
setmqaut -m JUPITER -t queue -n MOON.EUROPA -g VOYAGER +browse +get
```

allows the members of the group VOYAGER to browse messages on the queue MOON.EUROPA that is owned by the queue manager JUPITER. It allows the members to get messages from the queue as well. To revoke these authorities later, enter the following command:

```
setmqaut -m JUPITER -t queue -n MOON.EUROPA -g VOYAGER -browse -get
```

The command:

```
setmqaut -m JUPITER -t queue -n MOON.* -g VOYAGER +put
```

allows the members of the group VOYAGER to put messages on any queue with a name that commences with the characters `MOON.`. MOON.* is the name of a generic profile. A *generic profile* allows you to grant authorities for a set of objects using a single **setmqaut** command.

The control command **dspmqaut** is available to display the current authorities that a user or group has for a specified object. The control command **dmpmqaut** is also available to display the current authorities associated with generic profiles.

IBM i On IBM i, an administrator uses the CL command GRTMQMAUT to grant authorities and the CL command RVKMQMAUT to revoke authorities. Generic profiles can be used as well. For example, the CL command:

```
GRTMQMAUT MQMNAME(JUPITER) OBJTYPE(*Q) OBJ('MOON.*') USER(VOYAGER) AUT(*PUT)
```

provides the same function as the previous example of a **setmqaut** command; it allows the members of the group VOYAGER to put messages on any queue with a name that commences with the characters `MOON.`

IBM i The CL command DSPMQMAUT displays the current authorities that user or group has for a specified object. The CL commands WRKMQMAUT and WRKMQMAUTD are also available to work with the current authorities associated with objects and generic profiles.

If you do not want any authority checks, for example, in a test environment, you can disable the OAM.

Using PCF to access OAM commands

On IBM i IBM i, UNIX, Linux and Windows systems, you can use PCF commands to access OAM administration commands.

The PCF commands and their equivalent OAM commands are as follows:

Table 21. PCF commands and their equivalent OAM commands

PCF command	OAM command
Inquire Authority Records	dmpmqaut
Inquire Entity Authority	dspmqaut
Set Authority Record	setmqaut
Delete Authority Record	setmqaut with -remove option

The **`setmqaut`** and **`dmpmqaut`** commands are restricted to members of the mqm group. The equivalent PCF commands can be executed by users in any group who have been granted dsp and chg authorities on the queue manager.

For more information about using these commands, see Introduction to Programmable Command Formats.

Authority to work with IBM MQ objects on z/OS

On z/OS, there are seven categories of authority check associated with calls to the MQI. You must define certain RACF profiles and give appropriate access to these profiles. Use the *RESLEVEL* profile to control how many users IDs are checked.

The seven categories of authority check associated with calls to the MQI:

Connection security
The authority checks that are performed when an application connects to a queue manager

Queue security
The authority checks that are performed when an application opens a queue or deletes a permanent dynamic queue

Process security
The authority checks that are performed when an application opens a process object

Namelist security
The authority checks that are performed when an application opens a namelist object

Alternate user security
The authority checks that are performed when an application requests alternate user authority when opening an object

Context security
The authority checks that are performed when an application opens a queue and specifies that it intends to set or pass the context information in the messages it puts on the queue

Topic security
The authority checks that are performed when an application opens a topic

Each category of authority check is implemented in the same way that command security and command resource security are implemented. You must define certain RACF profiles and give the necessary groups and user IDs access to these profiles at the required levels. For queue security, the level of access determines the types of operation the application can perform on a queue. For context security, the level of access determines whether the application can:

- Pass all the context fields
- Pass all the context fields and set the identity context fields
- Pass and set all the context fields

Each category of authority check can be turned on or off by defining switch profiles.

All the categories, except connection security, are known collectively as *API-resource security*.

By default, when an API-resource security check is performed as a result of an MQI call from an application using a batch connection, only one user ID is checked. When a check is performed as a result of an MQI call from a CICS or IMS application, or from the channel initiator, two user IDs are checked.

By defining a *RESLEVEL profile*, however, you can control whether zero, one, or two users IDs are checked. The number of user IDs that are checked is determined by the user ID associated with the type of connection when an application connects to the queue manager and the access level that user ID has to the RESLEVEL profile. The user ID associated with each type of connection is:

- The user ID of the connecting task for batch connections
- The CICS address space user ID for CICS connections

- The IMS region address space user ID for IMS connections
- The channel initiator address space user ID for channel initiator connections

For more information about the authority to work with IBM MQ objects on z/OS, see “Authority to administer IBM MQ on z/OS” on page 450.

Security for remote messaging

This section deals with remote messaging aspects of security.

You must provide users with authority to use the IBM MQ facilities. This is organized according to actions to be taken with respect to objects and definitions. For example:

- Queue managers can be started and stopped by authorized users
- Applications must connect to the queue manager and have authority to use queues
- Message channels must be created and controlled by authorized users
- Objects are kept in libraries and access to these libraries can be restricted

The message channel agent at a remote site must check that the message being delivered originated from a user with authority to do so at this remote site. In addition, as MCAs can be started remotely, it might be necessary to verify that the remote processes trying to start your MCAs are authorized to do so. There are four possible ways for you to deal with this:

1. Make appropriate use of the PutAuthority attribute of your RCVR, RQSTR, or CLUSRCVR channel definition to control which user is used for authorization checks at the time incoming messages are put to your queues. See the DEFINE CHANNEL command description in the MQSC Command Reference.
2. Implement channel authentication records to reject unwanted connection attempts, or to set an MCAUSER value based on the following: the remote IP address, the remote user ID, the SSL or TLS Subject Distinguished Name (DN) provided, or the remote queue manager name.
3. Implement *user exit* security checking to ensure that the corresponding message channel is authorized. The security of the installation hosting the corresponding channel ensures that all users are properly authorized, so that you do not need to check individual messages.
4. Implement *user exit* message processing to ensure that individual messages are vetted for authorization.

Security of IBM MQ for IBM i objects

This section deals with remote messaging aspects of security.

You must provide users with authority to make use of the IBM MQ for IBM i facilities. This authority is organized according to actions to be taken with respect to objects and definitions. For example:

- Queue managers can be started and stopped by authorized users
- Applications need to connect to the queue manager, and have authority to make use of queues
- Message channels need to be created and controlled by authorized users

The message channel agent at a remote site must check that the message being delivered has derived from a user with authority to isue the message at this remote site. In addition, as MCAs can be started remotely, it might be necessary to verify that the remote processes trying to start your MCAs are authorized to do so. There are four possible ways for you to deal with this:

- Decree in the channel definition that messages must contain acceptable *context* authority, otherwise they are discarded.
- Implement channel authentication records to reject unwanted connection attempts, or to set an MCAUSER value based on one of the following: the remote IP address, the remote user ID, the SSL or TLS Distinguished Name (DN) provided, or the remote queue manager name.

- Implement user exit security checking to ensure that the corresponding message channel is authorized. The security of the installation hosting the corresponding channel ensures that all users are properly authorized, so that you do not need to check individual messages.
- Implement user exit message processing to ensure that individual messages are vetted for authorization.

Here are some facts about the way IBM MQ for IBM i operates security:
- Users are identified and authenticated by IBM i.
- Queue manager services invoked by applications are run with the authority of the queue manager user profile, but in the user's process.
- Queue manager services invoked by user commands are run with the authority of the queue manager user profile.

Security of objects on UNIX and Linux systems

Administration users must be part of the mqm group on your system (including root) if this ID is going to use IBM MQ administration commands.

You should always run amqcrsta as the "mqm" user ID.

User IDs on UNIX and Linux systems

The queue manager converts all uppercase or mixed case user identifiers into lowercase. The queue manager then inserts the user identifiers into the context part of a message, or checks their authorization. Authorizations are therefore based only on lowercase identifiers.

Security of objects on Windows systems

Administration users must be part of both the mqm group and the administrators group on Windows systems if this ID is going to use IBM MQ administration commands.

User IDs on Windows systems

On Windows systems, *if there is no message exit installed*, the queue manager converts any uppercase or mixed case user identifiers into lowercase. The queue manager then inserts the user identifiers into the context part of a message, or checks their authorization. Authorizations are therefore based only on lowercase identifiers.

User IDs across systems

Platforms other than Windows, UNIX and Linux systems use uppercase characters for user IDs in messages.

To allow Windows, UNIX and Linux systems to use lowercase user IDs in messages, the following conversions are carried out by the message channel agent (MCA) on these platforms:

At the sending end
The alphabetic characters in all user IDs are converted to uppercase characters, if there is no message exit installed.

At the receiving end
The alphabetic characters in all user IDs are converted to lowercase characters, if there is no message exit installed.

The automatic conversions are not carried out if you provide a message exit on UNIX, Linux and Windows systems for any other reason.

Using a custom authorization service

IBM MQ supplies an installable authorization service. You can choose to install an alternative service.

The authorization service component supplied with IBM MQ is called the Object Authority Manager (OAM). If the OAM does not supply the authorization facilities you need, you can write your own authorization service component. The installable service functions that must be implemented by an authorization service component are described at Installable services interface reference information.

Access control for clients

Access control is based on user IDs. There can be many user IDs to administer, and user IDs can be in different formats. You can set the server-connection channel property MCAUSER to a special user ID value for use by clients.

Access control in IBM MQ is based on user IDs. The user ID of the process making MQI calls is normally used. For MQ MQI clients, the server-connection MCA makes MQI calls on behalf of MQ MQI clients. You can select an alternative user ID for the server-connection MCA to use for making MQI calls. The alternative user ID can be associated either with the client workstation, or with anything you choose to organize and control the access of clients. The user ID needs to have the necessary authorities allocated to it on the server to issue MQI calls. Choosing an alternative user ID is preferable to allowing clients to make MQI calls with the authority of the server-connection MCA.

Table 22. The user ID used by a server-connection channel

User ID	**When used**
The user ID that is set by a security exit	Used unless blocked by a `CHLAUTH TYPE(BLOCKUSER)` rule. See the following section, "Setting the user ID in a security exit" on page 468 for more information.
The user ID that is set by a CHLAUTH rule	Used unless over-ridden by a security exit. See Channel Authentication Records for more information.
The user ID that is defined in the `MCAUSER` attribute in the SVRCONN channel definition	Used unless over-ridden by a security exit or a CHLAUTH rule.
The user ID that is flowed from the client machine	Used when no user ID is set by any other means.
The user ID that started the server-connection channel	Used when no user ID is set by any other means and no client user ID is flowed. See the following section, "The user ID that runs the channel program" on page 468 for more information.

Because the server-connection MCA makes MQI calls on behalf of remote users, it is important to consider the security implications of the server-connection MCA issuing MQI calls on behalf of remote clients and how to administer the access of a potentially large number of users.

- One approach is for the server-connection MCA to issue MQI calls on its own authority. But beware, it is normally undesirable for the server-connection MCA, with its powerful access capabilities, to issue MQI calls on behalf of client users.
- Another approach is to use the user ID that flows from the client. The server-connection MCA can issue MQI calls using the access capabilities of the client user ID. This approach presents a number of questions to consider:
  1. There are different formats for the user ID on different platforms. This sometimes causes problems if the format of the user ID on the client differs from the acceptable formats on the server.
  2. There are potentially many clients, with different, and changing user IDs. The IDs need to be defined and managed on the server.
  3. Is the user ID to be trusted? Any user ID can be flowed from a client, not necessarily the ID of the logged on user. For example, the client might flow an ID with full `mqm` authority that was intentionally only defined on the server for security reasons.
- The preferred approach is to define client identification tokens at the server, and so limit the capabilities of client connected applications. This is typically done by setting the server-connection channel property MCAUSER to a special user ID value to be used by clients, and defining few IDs for use by clients with different level of authorization on the server.

Setting the user ID in a security exit

For IBM MQ MQI clients, the process that issues the MQI calls is the server-connection MCA. The user ID used by the server-connection MCA is contained in either the `MCAUserIdentifier` or `LongMCAUserIdentifier` fields of the MQCD. The contents of these fields are set by:

- Any values set by security exits
- The user ID from the client
- MCAUSER (in the server-connection channel definition)

The security exit can override the values that are visible to it, when it is invoked.

- If the server-connection channel MCAUSER attribute is set to nonblank, the MCAUSER value is used.
- If the server-connection channel MCAUSER attribute is blank, the user ID received from the client is used.
- If the server-connection channel MCAUSER attribute is blank, and no user ID is received from the client then the user ID that started the server-connection channel is used.

The IBM MQ client does not flow the asserted user ID to the server when a client-side security exit is in use.

The user ID that runs the channel program

When the user ID fields are derived from the user ID that started the server-connection channel, the following value is used:

- For z/OS, the user ID assigned to the channel initiator started task by the z/OS started procedures table.
- For TCP/IP (non- z/OS), the user ID from the `inetd.conf` entry, or the user ID that started the listener.
- For SNA (non- z/OS), the user ID from the SNA Server entry or (if there is none) the incoming attach request, or the user ID that started the listener.
- For NetBIOS or SPX, the user ID that started the listener.

If any server-connection channel definitions exist that have the MCAUSER attribute set to blank, clients can use this channel definition to connect to the queue manager with access authority determined by the user ID supplied by the client. This might be a security exposure if the system on which the queue manager is running allows unauthorized network connections. The IBM MQ default server-connection channel (SYSTEM.DEF.SVRCONN) has the MCAUSER attribute set to blank. To prevent unauthorized access, update the MCAUSER attribute of the default definition with a user ID that has no access to IBM MQ MQ objects.

Case of user IDs

When you define a channel with `runmqsc`, the MCAUSER attribute is changed to uppercase unless the user ID is contained within single quotation marks.

For servers on UNIX, Linux and Windows systems, the content of the `MCAUserIdentifier` field that is received from the client is changed to lowercase.

For servers on IBM i, the content of the `LongMCAUserIdentifier` field that is received from the client is changed to uppercase.

For servers on UNIX and Linux systems, the content of the `LongMCAUserIdentifier` field that is received from the client is changed to lowercase.

By default, the user ID that is passed when a MQ JMS binding application is used, is the user ID for the JVM the application is running on.

It is also possible to pass a user ID via the `createQueueConnection` method.

Planning confidentiality

Plan how to keep your data confidential.

You can implement confidentiality at the application level or at link level. You might choose to use SSL or TLS, in which case you must plan your usage of digital certificates. You can also use channel exit programs if standard facilities do not satisfy your requirements.

Related concepts:

"Comparing link level security and application level security"
This topic contains information about various aspects of link level security and application level security, and compares the two levels of security.

"Channel exit programs" on page 476
Channel exit programs are programs that are called at defined places in the processing sequence of an MCA. Users and vendors can write their own channel exit programs. Some are supplied by IBM.

"Protecting channels with SSL and TLS" on page 487
SSL and TLS support in IBM MQ uses the queue manager authentication information object, and various MQSC commands. You must also consider your use of digital certificates.

Comparing link level security and application level security

This topic contains information about various aspects of link level security and application level security, and compares the two levels of security.

Link level and application level security are illustrated in Figure 63.

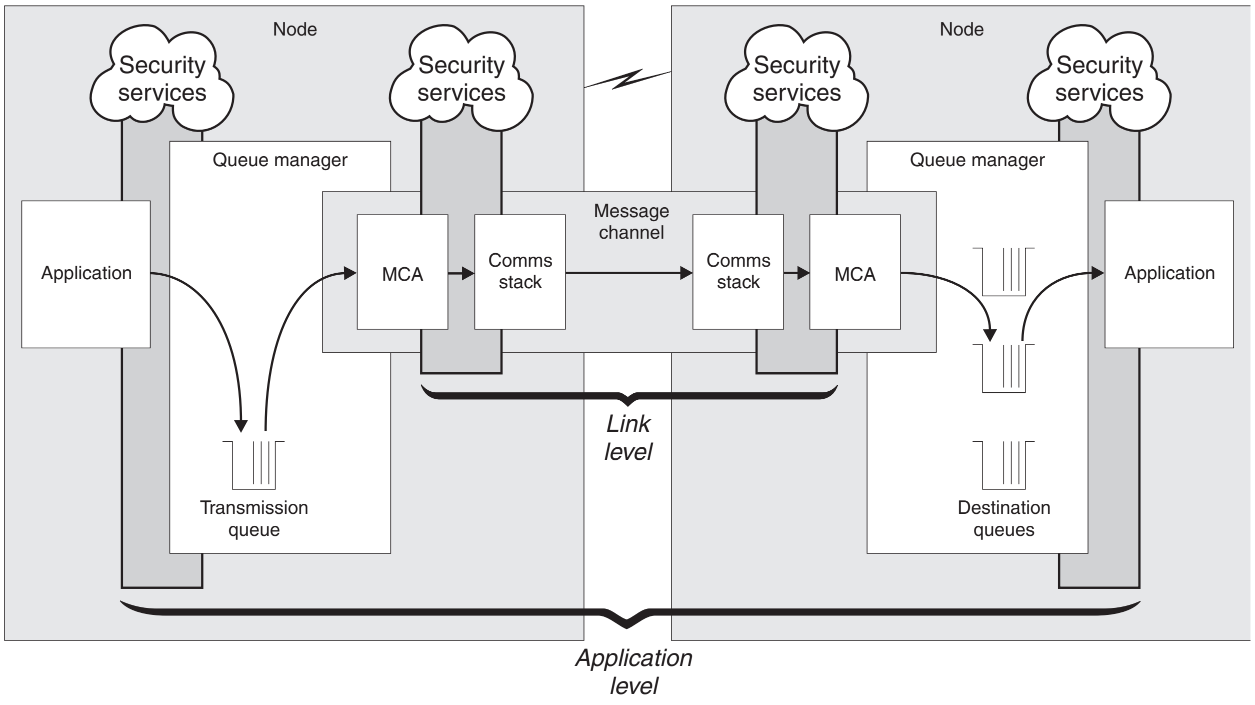


Figure 63. Link level security and application level security

Protecting messages in queues

Link level security can protect messages while they are transferred from one queue manager to another. It is particularly important when messages are transmitted over an insecure network. It cannot, however, protect messages while they are stored in queues at either a source queue manager, a destination queue manager, or an intermediate queue manager.

Application level security, by comparison, can protect messages while they are stored in queues and applies even when distributed queuing is not used. This is the major difference between link level security and application level security and is illustrated in Figure 63 on page 471.

Queue managers not running in controlled and trusted environments

If a queue manager is running in a controlled and trusted environment, the access control mechanisms provided by IBM MQ might be considered sufficient to protect the messages stored on its queues. This is particularly true if only local queuing is involved and messages never leave the queue manager. Application level security in this case might be considered unnecessary.

Application level security might also be considered unnecessary if messages are transferred to another queue manager that is also running in a controlled and trusted environment, or are received from such a queue manager. The need for application level security becomes greater when messages are transferred to, or received from, a queue manager that is not running in a controlled and trusted environment.

Differences in cost

Application level security might cost more than link level security in terms of administration and performance.

The cost of administration is likely to be greater because there are potentially more constraints to configure and maintain. For example, you might need to ensure that a particular user sends only certain types of message and sends messages only to certain destinations. Conversely, you might need to ensure that a particular user receives only certain types of message and receives messages only from certain sources. Instead of managing the link level security services on a single message channel, you might need to be configuring and maintaining rules for every pair of users who exchange messages across that channel.

There might be an effect on performance if security services are invoked every time an application puts or gets a message.

Organizations tend to consider link level security first because it might be easier to implement. They consider application level security if they discover that link level security does not satisfy all their requirements.

Availability of components

Generally, in a distributed environment, a security service requires a component on at least two systems. For example, a message might be encrypted on one system and decrypted on another. This applies to both link level security and application level security.

In a heterogeneous environment, with different platforms in use, each with different levels of security function, the required components of a security service might not be available for every platform on which they are needed and in a form that is easy to use. This is probably more of an issue for application level security than for link level security, particularly if you intend to provide your own application level security by buying in components from various sources.

Messages in a dead letter queue

If a message is protected by application level security, there might be a problem if, for any reason, the message does not reach its destination and is put on a dead letter queue. If you cannot work out how to process the message from the information in the message descriptor and the dead letter header, you might need to inspect the contents of the application data. You cannot do this if the application data is encrypted and only the intended recipient can decrypt it.

What application level security cannot do

Application level security is not a complete solution. Even if you implement application level security, you might still require some link level security services. For example:

- When a channel starts, the mutual authentication of the two MCAs might still be a requirement. This can be done only by a link level security service.
- Application level security cannot protect the transmission queue header, MQXQH, which includes the embedded message descriptor. Nor can it protect the data in IBM MQ channel protocol flows other than message data. Only link level security can provide this protection.
- If application level security services are invoked at the server end of an MQI channel, the services cannot protect the parameters of MQI calls that are sent over the channel. In particular, the application data in an MQPUT, MQPUT1, or MQGET call is unprotected. Only link level security can provide the protection in this case.

Link level security

Link level security refers to those security services that are invoked, directly or indirectly, by an MCA, the communications subsystem, or a combination of the two working together.

Link level security is illustrated in Figure 63 on page 471.

Here are some examples of link level security services:

- The MCA at each end of a message channel can authenticate its partner. This is done when the channel starts and a communications connection has been established, but before any messages start to flow. If authentication fails at either end, the channel is closed and no messages are transferred. This is an example of an identification and authentication service.
- A message can be encrypted at the sending end of a channel and decrypted at the receiving end. This is an example of a confidentiality service.
- A message can be checked at the receiving end of a channel to determine whether its contents have been deliberately modified while it was being transmitted over the network. This is an example of a data integrity service.

Link level security provided by IBM MQ

The primary means of provision of confidentiality and data integrity in IBM MQ is by the use of SSL or TLS. For more information about the use of SSL and TLS in IBM MQ, see “SSL and TLS security protocols in IBM MQ” on page 397. For authentication, IBM MQ provides the facility to use channel authentication records. Channel authentication records offer precise control over the access granted to connecting systems, at the level of individual channels or groups of channels. For more information, see “Channel authentication records” on page 424.

Providing your own link level security

This collection of topics describes how you can provide your own link level security services. Writing your own channel exit programs is the main way to provide your own link level security services.

Channel exit programs are introduced in "Channel exit programs" on page 476. The same topic also describes the channel exit program that is supplied with IBM MQ for Windows (the SSPI channel exit program). This channel exit program is supplied in source format so that you can modify the source code to suit your requirements. If this channel exit program, or channel exit programs available from other vendors, do not meet your requirements, you can design and write your own. This topic suggests ways in which channel exit programs can provide security services. For information about how to write a channel exit program, see Writing channel-exit programs.

Link level security using a security exit:

Security exits normally work in pairs; one at each end of a channel. They are called immediately after the initial data negotiation has completed on channel startup.

Security exits can be used to provide identification and authentication, access control, and confidentiality.

Link level security using a message exit:

A message exit can be used only on a message channel, not on an MQI channel. It has access to both the transmission queue header, MQXQH, which includes the embedded message descriptor, and the application data in a message. It can modify the contents of the message and change its length.

A message exit can be used for any purpose that requires access to the whole message rather than a portion of it.

Message exits can be used to provide identification and authentication, access control, confidentiality, data integrity, and non-repudiation, and for reasons other than security.

Link level security using send and receive exits:

Send and receive exits can be used on both message and MQI channels. They are called for all types of data that flow on a channel, and for flows in both directions.

Send and receive exits have access to each transmission segment. They can modify its contents and change its length.

On a message channel, if an MCA needs to split a message and send it in more than one transmission segment, a send exit is called for each transmission segment containing a portion of the message and, at the receiving end, a receive exit is called for each transmission segment. The same occurs on an MQI channel if the input or output parameters of an MQI call are too large to be sent in a single transmission segment.

On an MQI channel, byte 10 of a transmission segment identifies the MQI call, and indicates whether the transmission segment contains the input or output parameters of the call. Send and receive exits can examine this byte to determine whether the MQI call contains application data that might need to be protected.

When a send exit is called for the first time, to acquire and initialize any resources it needs, it can ask the MCA to reserve a specified amount of space in the buffer that holds a transmission segment. When it is called later to process a transmission segment, it can use this space to add an encrypted key or a digital signature, for example. The corresponding receive exit at the other end of the channel can remove the data added by the send exit, and use it to process the transmission segment.

Send and receive exits are best suited for purposes in which they do not need to understand the structure of the data they are handling and can therefore treat each transmission segment as a binary object.

Send and receive exits can be used to provide confidentiality and data integrity, and for uses other than security.

Related information:

Identifying the API call in a send or receive exit program

Application level security

Application level security refers to those security services that are invoked at the interface between an application and a queue manager to which it is connected.

These services are invoked when the application issues MQI calls to the queue manager. The services might be invoked, directly or indirectly, by the application, the queue manager, another product that supports IBM MQ, or a combination of any of these working together. Application level security is illustrated in Figure 63 on page 471.

Application level security is also known as *end-to-end security* or *message level security*.

Here are some examples of application level security services:

- When an application puts a message on a queue, the message descriptor contains a user ID associated with the application. However, there is no data present, such as an encrypted password, that can be used to authenticate the user ID. A security service can add this data. When the message is eventually retrieved by the receiving application, another component of the service can authenticate the user ID using the data that has travelled with the message. This is an example of an identification and authentication service.
- A message can be encrypted when it is put on a queue by an application and decrypted when it is retrieved by the receiving application. This is an example of a confidentiality service.
- A message can be checked when it is retrieved by the receiving application. This check determines whether its contents have been deliberately modified since it was first put on a queue by the sending application. This is an example of a data integrity service.

IBM MQ Advanced Message Security

IBM MQ Advanced Message Security (IBM MQ AMS) is a separately licensed component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

If you are moving highly sensitive or valuable information, especially confidential or payment-related information such as patient records or credit card details, you must pay special attention to information security. Ensuring that information moving around the enterprise retains its integrity and is protected from unauthorized access is an ongoing challenge and responsibility. You are also likely to be required to comply with security regulations, at the risk of penalties for non-compliance.

You can develop your own security extensions to IBM MQ. However, such solutions require specialist skills and can be complicated and expensive to maintain. IBM MQ Advanced Message Security helps address these challenges when moving information around the enterprise between virtually every type of commercial IT system.

IBM MQ Advanced Message Security extends the security features of IBM MQ in the following ways:

- It provides application-level, end-to-end data protection for your point to point messaging infrastructure, using either encryption or digital signing of messages.
- It provides comprehensive security without writing complex security code or modifying or recompiling existing applications.

- It uses Public Key Infrastructure (PKI) technology to provide authentication, authorization, confidentiality, and data integrity services for messages.
- It provides administration of security policies for mainframe and distributed servers.
- It supports both IBM MQ servers and clients.
- It integrates with IBM MQ Managed File Transfer to provide an end-to-end secure messaging solution.

For more information, see “IBM MQ Advanced Message Security” on page 849.

Providing your own application level security

This collection of topics describes how you can provide your own application level security services.

To help you implement application level security, IBM MQ provides two exits, the API exit and the API-crossing exit.

These exits can provide identification and authentication, access control, confidentiality, data integrity, and non-repudiation services, and other functions not related to security.

If the API exit or API-crossing exit is not supported in your system environment, you might want to consider other ways of providing your own application level security. One way is to develop a higher level API that encapsulates the MQI. Programmers then use this API, instead of the MQI, to write IBM MQ applications.

The most common reasons for using a higher level API are:
- To hide the more advanced features of the MQI from programmers.
- To enforce standards in the use of the MQI.
- To add function to the MQI. This additional function can be security services.

Some vendor products use this technique to provide application level security for IBM MQ.

If you are planning to provide security services in this way, note the following regarding data conversion:
- If a security token, such as a digital signature, has been added to the application data in a message, any code performing data conversion must be aware of the presence of this token.
- A security token might have been derived from a binary image of the application data. Therefore, any checking of the token must be done before converting the data.
- If the application data in a message has been encrypted, it must be decrypted before data conversion.

Channel exit programs

Channel exit programs are programs that are called at defined places in the processing sequence of an MCA. Users and vendors can write their own channel exit programs. Some are supplied by IBM.

There are several types of channel exit program, but only four have a role in providing link level security:
- Security exit
- Message exit
- Send exit
- Receive exit

These four types of channel exit program are illustrated in Figure 64 on page 477 and are described in the following topics.

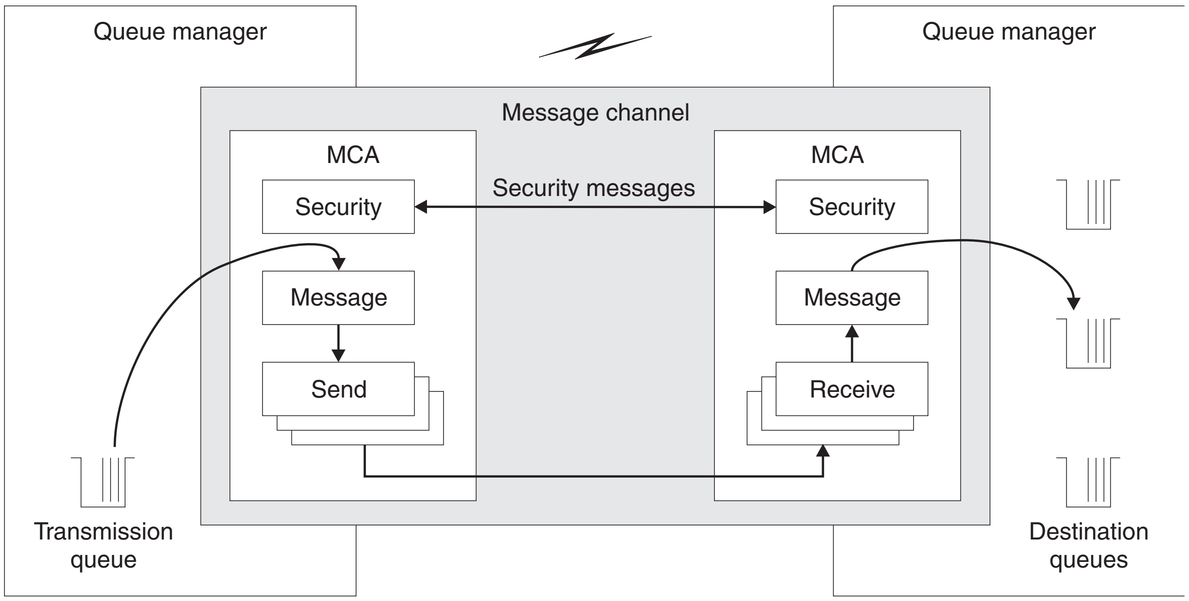


Figure 64. Security, message, send, and receive exits on a message channel

Related information:

Channel-exit programs for messaging channels

Security exit overview

Security exits normally work in pairs. They are called before messages flow and their purpose is to allow an MCA to authenticate its partner.

Security exits normally work in pairs; one at each end of a channel. They are called immediately after the initial data negotiation has completed on channel startup, but before any messages start to flow. The primary purpose of the security exit is to enable the MCA at each end of a channel to authenticate its partner. However, there is nothing to prevent a security exit from performing other function, even function that has nothing to do with security.

Security exits can communicate with each other by sending *security messages*. The format of a security message is not defined and is determined by the user. One possible outcome of the exchange of security messages is that one of the security exits might decide not to proceed any further. In that case, the channel is closed and messages do not flow. If there is a security exit at only one end of a channel, the exit is still called and can elect whether to continue or to close the channel.

Security exits can be called on both message and MQI channels. The name of a security exit is specified as a parameter in the channel definition at each end of a channel.

For more information about security exits, see “Link level security using a security exit” on page 474.

Message exit

Message exits operate only on message channels and normally work in pairs. A message exit can operate on the whole message and make various changes to it.

Message exits at the sending and receiving ends of a channel normally work in pairs. A message exit at the sending end of a channel is called after the MCA has got a message from the transmission queue. At the receiving end of a channel, a message exit is called before the MCA puts a message on its destination queue.

A message exit has access to both the transmission queue header, MQXQH, which includes the embedded message descriptor, and the application data in a message. A message exit can modify the contents of the message and change its length. A change of length might be the result of compressing, decompressing, encrypting, or decrypting the message. It might also be the result of adding data to the message, or removing data from it.

Message exits can be used for any purpose that requires access to the whole message, rather than a portion of it, and not necessarily for security.

A message exit can determine that the message it is currently processing should not proceed any further towards its destination. The MCA then puts the message on the dead letter queue. A message exit can also close the channel.

Message exits can be called only on message channels, not on MQI channels. This is because the purpose of an MQI channel is to enable the input and output parameters of MQI calls to flow between the IBM MQ MQI client application and the queue manager.

The name of a message exit is specified as a parameter in the channel definition at each end of a channel. You can also specify a list of message exits to be run in succession.

For more information about message exits, see “Link level security using a message exit” on page 474.

Send and receive exits

Send and receive exits typically work in pairs. They operate on transmission segments and are best used where the structure of the data they are processing is not relevant.

A *send exit* at one end of a channel and a *receive exit* at the other end normally work in pairs. A send exit is called just before an MCA issues a communications send to send data over a communications connection. A receive exit is called just after an MCA has regained control following a communications receive and has received data from a communications connection. If sharing conversations is in use, over an MQI channel, a different instance of a send and receive exit is called for each conversation.

The IBM MQ channel protocol flows between two MCAs on a message channel contain control information as well as message data. Similarly, on an MQI channel, the flows contain control information as well as the parameters of MQI calls. Send and receive exits are called for all types of data.

Message data flows in only one direction on a message channel but, on an MQI channel, the input parameters of an MQI call flow in one direction and the output parameters flow in the other. On both message and MQI channels, control information flows in both directions. As a result, send and receive exits can be called at both ends of a channel.

The unit of data that is transmitted in a single flow between two MCAs is called a *transmission segment*. Send and receive exits have access to each transmission segment. They can modify its contents and change its length. A send exit, however, must not change the first 8 bytes of a transmission segment. These 8 bytes form part of the IBM MQ channel protocol header. There are also restrictions on how much a send exit can increase the length of a transmission segment. In particular, a send exit cannot increase its length beyond the maximum that was negotiated between the two MCAs at channel startup.

On a message channel, if a message is too large to be sent in a single transmission segment, the sending MCA splits the message and sends it in more than one transmission segment. As a consequence, a send exit is called for each transmission segment containing a portion of the message and, at the receiving end, a receive exit is called for each transmission segment. The receiving MCA reconstitutes the message from the transmission segments after they have been processed by the receive exit.

Similarly, on an MQI channel, the input or output parameters of an MQI call are sent in more than one transmission segment if they are too large. This might occur, for example, on an MQPUT, MQPUT1, or MQGET call if the application data is sufficiently large.

Taking these considerations into account, it is more appropriate to use send and receive exits for purposes in which they do not need to understand the structure of the data they are handling and can therefore treat each transmission segment as a binary object.

A send or a receive exit can close a channel.

The names of a send exit and a receive exit are specified as parameters in the channel definition at each end of a channel. You can also specify a list of send exits to be run in succession. Similarly, you can specify a list of receive exits.

For more information about send and receive exits, see “Link level security using send and receive exits” on page 474.

Planning data integrity

Plan how to preserve the integrity of your data.

You can implement data integrity at the application level or at link level.

At the application level, you can use API exit programs if standard facilities do not satisfy your requirements. You might choose to use IBM MQ Advanced Message Security (AMS) to digitally sign messages in order to protect against unauthorized modification.

At the link level, you might choose to use SSL or TLS, in which case you must plan your usage of digital certificates. You can also use channel exit programs if standard facilities do not satisfy your requirements.

Related concepts:

"Protecting channels with SSL and TLS" on page 487
SSL and TLS support in IBM MQ uses the queue manager authentication information object, and various MQSC commands. You must also consider your use of digital certificates.

"Data integrity in IBM MQ" on page 396
You can use a data integrity service to detect whether a message has been modified.

"IBM MQ Advanced Message Security" on page 475
IBM MQ Advanced Message Security (IBM MQ AMS) is a separately licensed component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

Related information:

API exit reference

Channel-exit calls and data structures

Planning auditing

Decide what data you need to audit, and how you will capture and process audit information. Consider how to check that your system is correctly configured.

There are several aspects to activity monitoring. The aspects you must consider are often defined by auditor requirements, and these requirements are often driven by regulatory standards such as HIPAA (Health Insurance Portability and Accountability Act) or SOX (Sarbanes-Oxley). IBM MQ provides features intended to help with compliance to such standards.

Consider whether you are interested only in exceptions or whether you are interested in all system behavior.

Some aspects of auditing can also be considered as operational monitoring; one distinction for auditing is that you are often looking at historic data, not just looking at real-time alerts. Monitoring is covered in the section Monitoring and performance.

What data to audit

Consider what types of data or activity you need to audit, as described in the following sections:

Changes made to IBM MQ using the IBM MQ interfaces
Configure IBM MQ to issue instrumentation events, specifically command events and configuration events.

Changes made to IBM MQ outside its control
Some changes can affect how IBM MQ behaves, but cannot be directly monitored by IBM MQ. Examples of such changes include changes to the configuration files `mqs.ini`, `qm.ini`, and `mqclient.ini`, the creation and deletion of queue managers, installation of binary files such as user exit programs, and changes to file permissions. To monitor these activities, you must use tools running at the level of the operating system. Different tools are available and appropriate for different operating systems. You might also have logs created by associated tools such as *sudo*.

Operational control of IBM MQ
You might have to use operating system tools to audit activities such as the starting and stopping of queue managers. In some cases, IBM MQ can be configured to issue instrumentation events.

Application activity within IBM MQ
To audit the actions of applications, for example opening of queues and putting and getting of messages, configure IBM MQ to issue appropriate events.

Intruder alerts
To audit attempted breaches of security, configure your system to issue authorization events. Channel events might also be useful to show activity, particularly if a channel ends unexpectedly.

Planning the capture, display, and archiving of audit data

Many of the elements you need are reported as IBM MQ event messages. You must choose tools that can read and format these messages. If you are interested in long-term storage and analysis you must move them to an auxiliary storage mechanism such as a database. If you do not process these messages, they remain on the event queue, possibly filling the queue. You might decide to implement a tool that automatically takes action based on some events; for example, to issue an alert when a security failure happens.

Verifying that your system is correctly configured

A set of tests are supplied with the MQ Explorer. Use these to check your object definitions for problems.

Also, check periodically that the system configuration is as you expect. Although command and configuration events can report when something is changed, it is also useful to dump the configuration and compare it to a known good copy.

Planning security by topology

This section covers security in specific situations, namely for channels, queue manager clusters, publish/subscribe and multicast applications, and when using a firewall.

See the following subtopics for more information:

Channel authorization

When you send or receive a message through a channel, you need to provide access to various IBM MQ resources. Message Channel Agents (MCAs) are essentially IBM MQ applications that move messages between queue managers, and as such require access to various IBM MQ resources to operate correctly.

To receive messages at PUT time for MCAs, you can use either the user ID associated with the MCA, or the user ID associated with the message.

At CONNECT time you can map the asserted user ID to an alternative user, by using **CHLAUTH** channel authentication records.

In IBM MQ, channels can be protected by TLS support.

The user IDs associated with sending and receiving channels, excluding the sender channel where the MCAUSER attribute is unused, require access to the following resources:

- The user ID associated with a sending channel requires access to the queue manager, the transmission queue, the dead-letter queue, and access to any other resources that are required by channel exits.
- The MCAUSER user ID of a receiver channel needs *+setall* authority. The reason is that the receiver channel has to create the full MQMD, including all context fields, using the data it received from the remote sender channel.

  The queue manager therefore requires that the user performing this activity has the *+setall* authority. This *+setall* authority must be granted to the user for:
  - All queues that the receiver channel validly puts messages to.
  - The queue manager object. See Authorizations for context for further information.
- The MCAUSER user ID of a receiver channel where the originator requested a COA report message needs *+passid* authority on the transmission queue that returns the report message. Without this authority, AMQ8077 error messages are logged.
- With the user ID associated with the receiving channel you can open the target queues to put messages onto the queues.

  This involves the Message queuing Interface (MQI), so additional access control checks might need to be made if you are not using the IBM MQ Object Authority Manager (OAM).

  You can specify whether the authorization checks are made against the user ID associated with the MCA (as described in this topic), or against the user ID associated with the message (from the MQMD UserIdentifier field).

  For the channel types to which it applies, the **PUTAUT** parameter of a channel definition specifies which user ID is used for these checks.
  - The channel defaults to using the queue manager's service account, that will have full administrative rights and requires no special authorizations

    In the case of server-connection channels, administrative connections are blocked by default by CHLAUTH rules and require explicit provisioning.

Channels of type receiver, requester, and cluster-receiver allow local administration by any adjacent queue manager, unless the administrator takes steps to restrict this access.

- V 8.0.0.4 Before Version 8.0.0, Fix Pack 4, if you use a user ID that lacks IBM MQ administrative privileges, then you must grant **dsp** and **ctrlx** authority for the channel to that user ID for the channel to work. From Version 8.0.0, Fix Pack 4, it is not necessary to grant *dsp* and *ctrlx* authority for the MCAUSER user ID of a receiver channel.

  Attention: When a channel reset is needed for message batch confirmation, IBM MQ attempts to query the channel, which does require **+dsp** authority.
- The MCAUSER attribute is unused for the SDR channel type.
- If you use the user ID associated with the message, it is likely that the user ID is from a remote system.

  This remote system user ID must be recognized by the target system. For example, issue the following commands:
  -
```
setmqaut -m QMgrName -t qmgr -g GroupName +connect +inq +setall
```
  -
```
setmqaut -m QMgrName -t chl -n Profile -g GroupName +dsp +ctrlx
```
    where *Profile* is a channel.
  -
```
setmqaut -m QMgrName -t q -n Profile -g GroupName +put +setall
```
    where *Profile* is a dead-letter queue, if set.
  -
```
setmqaut -m QMgrName -t q -n Profile -g GroupName +put +setall
```
    where *Profile* is a list of authorized queues.

  Attention: Exercise caution when authorizing a user ID to place messages onto the Command Queue or other sensitive system queues.

The user ID associated with the MCA depends on the type of MCA. There are two types of MCA:

Caller MCA
MCAs that initiate a channel. Caller MCAs can be started as individual processes, as threads of the channel initiator, or as threads of a process pool. The user ID used is the user ID associated with the parent process (the channel initiator), or the user ID associated with the process that starts the MCA.

Responder MCA
Responder MCAs are MCAs that are started as a result of a request by a caller MCA. Responder MCAs can be started as individual processes, as threads of the listener, or as threads of a process pool. The user ID can be any one of the following types (in this order of preference):

1. On APPC, the caller MCA can indicate the user ID to be used for the responder MCA. This is called the network user ID and applies only to channels started as individual processes. Set the network user ID by using the USERID parameter of the channel definition.
2. If the **USERID** parameter is not used, the channel definition of the responder MCA can specify the user ID that the MCA must use. Set the user ID by using the **MCAUSER** parameter of the channel definition.
3. If the user ID has not been set by either of the previous (two) methods, the user ID of the process that starts the MCA or the user ID of the parent process (the listener) is used.

Related concepts:

"Channel authentication records" on page 424
To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records.

Related information:

Channel authentication record properties

Protecting channel initiator definitions

Only members of the mqm group can manipulate channel initiators.

IBM MQ channel initiators are not IBM MQ objects; access to them is not controlled by the OAM. IBM MQ does not allow users or applications to manipulate these objects, unless their user ID is a member of the mqm group. If you have an application that issues the PCF command **StartChannelInitiator** , the user ID specified in the message descriptor of the PCF message must be a member of the mqm group on the target queue manager.

A user ID must also be a member of the mqm group on the target machine to issue the equivalent MQSC commands through the Escape PCF command or using **runmqsc** in indirect mode.

Transmission queues

Queue managers automatically put remote messages on a transmission queue; no special authority is required for this.

However, if you need to put a message directly on a transmission queue, this requires special authorization; see Table 25 on page 521.

Channel exits

If channel authentication records are not suitable, you can use channel exits for added security. A security exit forms a secure connection between two security exit programs. One program is for the sending message channel agent (MCA), and one is for the receiving MCA.

See "Channel exit programs" on page 476 for more information about channel exits.

Protecting channels with SSL and TLS

SSL and TLS support in IBM MQ uses the queue manager authentication information object, and various MQSC commands. You must also consider your use of digital certificates.

Digital certificates and key repositories

It is good practice to set the queue manager certificate label attribute (`CERTLABL`) to the name of the personal certificate to be used for the majority of channels, and override it for exceptions, by setting the certificate label on those channels that require different certificates.

If you need many channels with certificates that differ from the default certificate set on the queue manager, you should consider dividing the channels between several queue managers or use an MQIPT proxy in front of the queue manager to present a different certificate.

You can use a different certificate for every channel, but if you store too many certificates in a key repository, you might expect performance to be affected when starting SSL/TLS channels. Try to keep the number of certificates in a key repository to less than about 50 and consider 100 to be a maximum as GSKit performance decreases sharply with larger key repositories.

Allowing multiple certificates on the same queue manager increases the chances that multiple CA certificates will be used on the same queue manager. This increases the odds of certificate Subject Distinguished Name namespace clashes for certificates issued by separate certificate authorities.

While professional certificate authorities are likely to be more careful, inhouse certificate authorities often lack clear naming conventions and you could end up with unintended matches between one CA and another.

You should check the certificate Issuer Distinguished Name in addition to the Subject Distinguished Name. To do this, use a channel authentication SSLPEERMAP record and set both the **SSLPEER** and **SSLCERTI** fields to match the Subject DN and Issuer DN respectively.

Self-signed and CA-signed certificates

It is important to plan your use of digital certificates, both when you are developing and testing your application, and for its use in production. You can use CA-signed certificates or self-signed certificates, depending on the usage of your queue managers and client applications.

CA-signed certificates
: For production systems, obtain your certificates from a trusted certificate authority (CA). When you obtain a certificate from an external CA, you pay for the service.

Self-signed certificates
: While you are developing your application you can use self-signed certificates or certificates issued by a local CA, depending on platform:

Windows UNIX Linux On Windows, UNIX, and Linux systems, you can use self-signed certificates. See “Creating a self-signed personal certificate on UNIX, Linux, and Windows systems” on page 681for instructions.

IBM i On IBM i systems, you can use certificates signed by the local CA. See “Requesting a server certificate on IBM i” on page 666 for instructions.

z/OS On z/OS, you can use either self-signed or local CA-signed certificates. See “Creating a self-signed personal certificate on z/OS” on page 706 or “Requesting a personal certificate on z/OS” on page 707 for instructions.

Self-signed certificates are not suitable for production use, for the following reasons:

- Self-signed certificates cannot be revoked, which might allow an attacker to spoof an identity after a private key has been compromised. CAs can revoke a compromised certificate, which prevents its further use. CA-signed certificates are therefore safer to use in a production environment, though self-signed certificates are more convenient for a test system.
- Self-signed certificates never expire. This is both convenient and safe in a test environment, but in a production environment it leaves them open to eventual security breaches. The risk is compounded by the fact that self-signed certificates cannot be revoked.
- A self-signed certificate is used both as a personal certificate and as a root (or trust anchor) CA certificate. A user with a self-signed personal certificate might be able to use it to sign other personal certificates. In general, this is not true of personal certificates issued by a CA, and represents a significant exposure.

CipherSpecs and digital certificates

Only a subset of the supported CipherSpecs can be used with all of the supported types of digital certificate. It is therefore necessary to choose an appropriate CipherSpec for your digital certificates. Similarly, if your organization's security policy requires that a particular CipherSpec be used, then you must obtain suitable digital certificates.

For more information on the relationship between CipherSpecs and digital certificates, refer to "Digital certificates and CipherSpec compatibility in IBM MQ" on page 419

Certificate validation policies

The IETF RFC 5280 standard specifies a series of certificate validation rules which compliant application software must implement in order to prevent impersonation attacks. A set of certificate validation rules is known as a certificate validation policy. For more information about certificate validation policies in IBM MQ, see "Certificate validation policies in IBM MQ" on page 418.

Planning for certificate revocation checking

Allowing multiple certificates from different certificate authorities potentially causes unnecessary additional certificate revocation checking.

In particular, if you have explicitly configured the use of a revocation server from a particular CA, for example by using an AUTHINFO object or Authentication information record (MQAIR) structure, a revocation check fails when presented with a certificate from a different CA.

You should avoid explicit certificate revocation server configuration. Instead, you should enable implicit checking where each certificate contains its own revocation server location in a certificate extension, for example, CRL Distribution Point, or OCSP AuthorityInfoAccess.

For more information, see OCSPCheckExtensions and CDPCheckExtensions.

Commands and attributes for SSL and TLS support

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols provide channel security, with protection against eavesdropping, tampering, and impersonation. IBM MQ support for SSL and TLS enables you to specify, on the channel definition, that a particular channel uses SSL or TLS security. You can also specify details of the type of security you want, such as the encryption algorithm you want to use.

- The following MQSC commands support SSL and TLS:

  ALTER AUTHINFO
  Modifies the attributes of an authentication information object.

  DEFINE AUTHINFO
  Creates an authentication information object.

  DELETE AUTHINFO
  Deletes an authentication information object.

  DISPLAY AUTHINFO
  Displays the attributes for a specific authentication information object.

- The following queue manager parameters support SSL and TLS:

  CERTLABL
  Defines a personal certificate label to use.

  SSLCRLNL
  The SSLCRLNL attribute specifies a namelist of authentication information objects which are used to provide certificate revocation locations to allow enhanced TLS/SSL certificate checking.

  SSLCRYP
  On Windows , UNIX and Linux systems, sets the **SSLCryptoHardware** queue manager attribute. This attribute is the name of the parameter string that you can use to configure the cryptographic hardware you have on your system.

SSLEV
: Determines whether an SSL/TLS event message is reported if a channel using SSL/TLS fails to establish an SSL/TLS connection.

SSLFIPS
: Specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM MQ , rather than in cryptographic hardware. If cryptographic hardware is configured, the cryptographic modules provided by the hardware product are used, and these might be FIPS-certified to a particular level. This depends on the hardware product in use.

SSLKEYR
: On Windows, UNIX and Linux systems, associates a key repository with a queue manager. The key database is held in a *GSKit* key database. The IBM Global Security Kit (GSKit) enables you to use SSL security on Windows , UNIX and Linux systems.

SSLRKEYC
: The number of bytes to be sent and received within an SSL/TLS conversation before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

- The following channel parameters support SSL and TLS:

CERTLABL
: Defines a personal certificate label to use.

SSLCAUTH
: Defines whether IBM MQ requires and validates a certificate from the SSL/TLS client.

SSLCIPH
: Specifies the encryption strength and function (CipherSpec), for example TLS_RSA_WITH_AES_128_CBC_SHA. The CipherSpec must match at both ends of channel.

SSLPEER
: Specifies the distinguished name (unique identifier) of allowed partners.

This section describes the **setmqaut** , **dspmqaut** , **dmpmqaut** , **rcrmqobj** , **rcdmqimg** , and **dspmqfls** commands to support the authentication information object. It also describes the iKeycmd command for managing certificates on UNIX and Linux systems, and the runmqakm tool for managing certificates on UNIX, Linux and Windows systems. See the following sections:

- setmqaut
- dspmqaut
- dmpmqaut
- rcrmqobj
- rcdmqimg
- dspmqfls
- Managing keys and certificates

For an overview of channel security using SSL/TLS, see

- "SSL and TLS security protocols in IBM MQ" on page 397

For details of MQSC commands associated with SSL/TLS, see

- ALTER AUTHINFO
- DEFINE AUTHINFO
- DELETE AUTHINFO
- DISPLAY AUTHINFO

For details of PCF commands associated with SSL/TLS, see

- Change, Copy, and Create Authentication Information Object

- Delete Authentication Information Object
- Inquire Authentication Information Object

IBM MQ for z/OS server connection channel

The IBM MQ for z/OS SVRCONN channel is not secure without implementing channel authentication, or adding a security exit using SSL or TLS. SVRCONN channels do not have a security exit defined by default.

Security concerns

SVRCONN channels are not secure as initially defined, SYSTEM.DEF.SVRCONN for example. To secure a SVRCONN channel you must set up channel authentication using the SET CHLAUTH command, or install a security exit and implement SSL, or TLS.

You must use a publicly available sample security exit, write a security exit yourself, or purchase a security exit.

There are several samples available that you can use as a good starting point for writing your own SVRCONN channel security exit.

In IBM MQ for z/OS, the member CSQ4BCX3 in your hlq.SCSQC37S library is a security exit sample written in the C language. Sample CSQ4BCX3 is also shipped pre-compiled in your hlq.SCSQAUTH library.

You can implement the CSQ4BCX3 sample exit by copying the compiled member hlq.SCSQAUTH(CSQ4BCX3) into a load library that is allocated to the CSQXLIB DD in your CHIN Proc. Note that the CHIN requires the load library to be set as "Program Controlled".

Alter your SVRCONN channel to set CSQ4BCX3 as the security exit.

When a client connects using that SVRCONN channel, CSQ4BCX3 will authenticate using the RemoteUserIdentifier and RemotePassword pair from MQCD. If authentication is successful it will copy RemoteUserIdentifier into MCAUserIdentifier, changing the identity context of the thread.

If you are writing an MQ Java client you can use pop-ups to query the user and set MQEnvironment.userID and MQEnvironment.password. These values will be passed when the connection is made.

Now that you have a functional security exit, there is the additional concern that the userid and password are being transmitted in plain text across the network when the connection is made, as are the contents of any subsequent MQ messages. You can use SSL to encrypt this initial connection information as well as the contents of any MQ messages.

Example

To secure the MQ Explorer SVRCONN channel SYSTEM.ADMIN.SVRCONN complete the following steps:

1. Copy hlq.SCSQAUTH(CSQ4BCX3) into a load library that is allocated to the CSQXLIB DD in the CHINIT Proc.
2. Verify that load library is Program Controlled.
3. Alter the SYSTEM ADMIN.SVRCONN to use security exit CSQ4BCX3.
4. In MQ Explorer, right-click the z/OS Queue Manager name, select **Connection Details** > **Properties** > **Userid** and enter your z/OS user ID.
5. Connect to the z/OS Queue Manager by entering a password.

Additional information

For exit CSQ4BCX3 to run in a Program Controlled environment, everything loaded into the CHIN address space must be loaded from a Program Controlled library, for example, all libraries in STEPLIB and any libraries named on CSQXLIB DD. To set a load library as Program Controlled issue RACF commands. In the following example the load library name is MY.TEST.LOADLIB.

```
RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
SETROPTS WHEN(PROGRAM)REFRESH
```

To alter the SVRCONN channel to implement CSQ4BCX3, issue the following MQ command:

```
ALTER CHANNEL( SYSTEM ADMIN.SVRCONN) CHLTYPE(SVRCONN) SECYEXIT(CSQ4BCX3)
```

In the example above, the SVRCONN channel name being used is SYSTEM ADMIN.SVRCONN.

See "Channel exit programs" on page 476 for more information about channel exits.

Related information:

Writing channel exit programs on z/OS

SNA LU 6.2 security services

SNA LU 6.2 offers session level cryptography, session level authentication, and conversation level authentication.

Note: This collection of topics assumes that you have a basic understanding of Systems Network Architecture (SNA). The other documentation referred to in this section contains a brief introduction to the relevant concepts and terminology. If you require a more comprehensive technical introduction to SNA, see *Systems Network Architecture Technical Overview*, GC30-3073.

SNA LU 6.2 provides three security services:

- Session level cryptography
- Session level authentication
- Conversation level authentication

For session level cryptography and session level authentication, SNA uses the *Data Encryption Standard (DES)* algorithm. The DES algorithm is a block cipher algorithm, which uses a symmetric key for encrypting and decrypting data. Both the block and the key are 8 bytes in length.

Session level cryptography

Session level cryptography encrypts and decrypts session data using the DES algorithm. It can therefore be used to provide a link level confidentiality service on SNA LU 6.2 channels.

Logical units (LUs) can provide mandatory (or required) data cryptography, selective data cryptography, or no data cryptography.

On a *mandatory cryptographic session*, an LU encrypts all outbound data request units and decrypts all inbound data request units.

On a *selective cryptographic session*, an LU encrypts only the data request units specified by the sending transaction program (TP). The sending LU signals that the data is encrypted by setting an indicator in the request header. By checking this indicator, the receiving LU can tell which request units to decrypt before passing them on to the receiving TP.

In an SNA network, IBM MQ MCAs are transaction programs. MCAs do not request encryption for any data that they send. Selective data cryptography is not an option therefore; only mandatory data cryptography or no data cryptography is possible on a session.

For information about how to implement mandatory data cryptography, see the documentation for your SNA subsystem. Refer to the same documentation for information about stronger forms of encryption that might be available for use on your platform, such as Triple DES 24-byte encryption on z/OS.

For more general information about session level cryptography, see *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808.

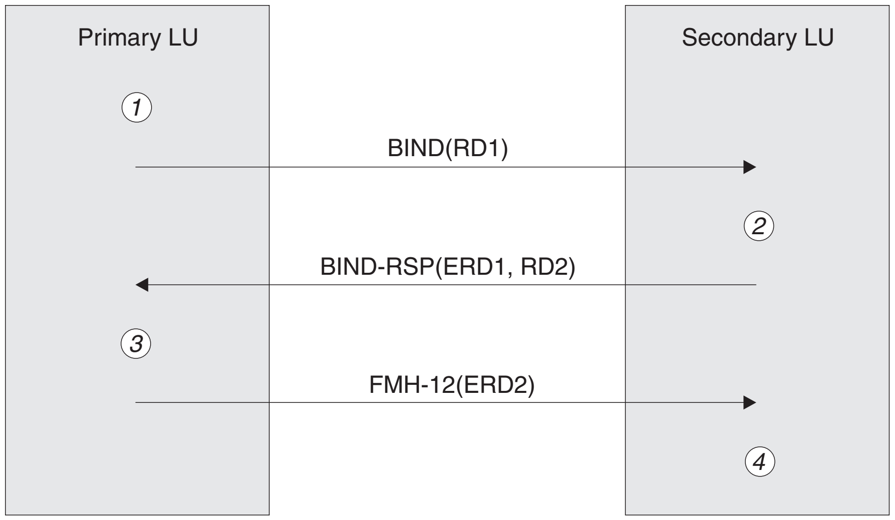
Session level authentication

Session level authentication is a session level security protocol that enables two LUs to authenticate each other while they are activating a session. It is also known as *LU-LU verification*.

Because an LU is effectively the "gateway" into a system from the network, you might consider this level of authentication to be sufficient in certain circumstances. For example, if your queue manager needs to exchange messages with a remote queue manager that is running in a controlled and trusted environment, you might be prepared to trust the identities of the remaining components of the remote system after the LU has been authenticated.

Session level authentication is achieved by each LU verifying its partner's password. The password is called an *LU-LU password* because one password is established between each pair of LUs. The way that an LU-LU password is established is implementation dependent and outside the scope of SNA.

Figure 65 illustrates the flows for session level authentication.



Legend:

BIND	=	BIND request unit
BIND-RSP	=	BIND response unit
ERD	=	Encrypted random data
FMH-12	=	Function Management Header 12
RD	=	Random data

Figure 65. Flows for session level authentication

The protocol for session level authentication is as follows. The numbers in the procedure correspond to the numbers in Figure 65.

1. The primary LU generates a random data value (RD1) and sends it to the secondary LU in the BIND request.

2. When the secondary LU receives the BIND request with the random data, it encrypts the data using the DES algorithm with its copy of the LU-LU password as the key. The secondary LU then generates a second random data value (RD2) and sends it, with the encrypted data (ERD1), to the primary LU in the BIND response.
3. When the primary LU receives the BIND response, it computes its own version of the encrypted data from the random data it generated originally. It does this by using the DES algorithm with its copy of the LU-LU password as the key. It then compares its version with the encrypted data that it received in the BIND response. If the two values are the same, the primary LU knows that the secondary LU has the same password as it does and the secondary LU is authenticated. If the two values do not match, the primary LU terminates the session.

   The primary LU then encrypts the random data that it received in the BIND response and sends the encrypted data (ERD2) to the secondary LU in a Function Management Header 12 (FMH-12).
4. When the secondary LU receives the FMH-12, it computes its own version of the encrypted data from the random data it generated. It then compares its version with the encrypted data that it received in the FMH-12. If the two values are the same, the primary LU is authenticated. If the two values do not match, the secondary LU terminates the session.

In an enhanced version of the protocol, which provides better protection against man in the middle attacks, the secondary LU computes a DES Message Authentication Code (MAC) from RD1, RD2, and the fully qualified name of the secondary LU, using its copy of the LU-LU password as the key. The secondary LU sends the MAC to the primary LU in the BIND response instead of ERD1.

The primary LU authenticates the secondary LU by computing its own version of the MAC, which it compares with the MAC received in the BIND response. The primary LU then computes a second MAC from RD1 and RD2, and sends the MAC to the secondary LU in the FMH-12 instead of ERD2.

The secondary LU authenticates the primary LU by computing its own version of the second MAC, which it compares with the MAC received in the FMH-12.

For information about how to configure session level authentication, see the documentation for your SNA subsystem. For more general information about session level authentication, see *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808.

Conversation level authentication

When a local TP attempts to allocate a conversation with a partner TP, the local LU sends an attach request to the partner LU, asking it to attach the partner TP. Under certain circumstances, the attach request can contain security information, which the partner LU can use to authenticate the local TP. This is known as *conversation level authentication*, or *end user verification*.

The following topics describe how IBM MQ provides support for conversation level authentication.

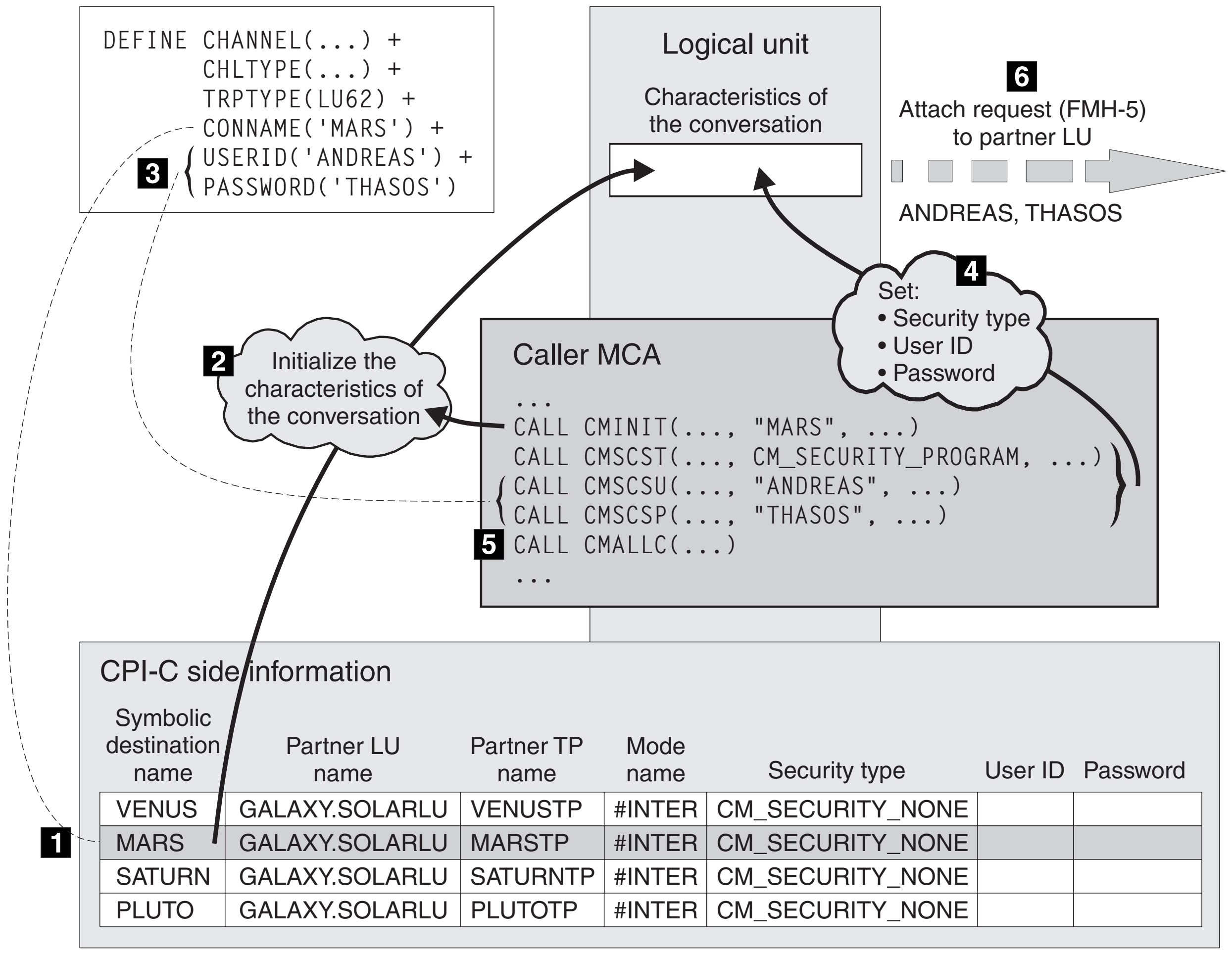
For more information about conversation level authentication, see *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808. For information specific to z/OS, see *z/OS MVS Planning: APPC/MVS Management*, SA22-7599.

For more information about CPI-C, see *Common Programming Interface Communications CPI-C Specification*, SC31-6180. For more information about APPC/MVS TP Conversation Callable Services, see *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*, SA22-7621.

Support for conversation level authentication in IBM MQ on : IBM i,UNIX systemsWindows

Use this topic to gain an overview of how conversation level authentication works, on platforms other than z/OS.

The support for conversation level authentication in IBM MQ for IBM i, IBM MQ on UNIX systems, and IBM MQ for Windows is illustrated in Figure 66. The numbers in the diagram correspond to the numbers in the description that follows.



```
DEFINE CHANNEL(...) +
       CHLTYPE(...) +
       TRPTYPE(LU62) +
       CONNAME('MARS') +
     3 USERID('ANDREAS') +
       PASSWORD('THASOS')
```

```
Caller MCA
...
CALL CMINIT(..., "MARS", ...)
CALL CMSCST(..., CM_SECURITY_PROGRAM, ...)
CALL CMSCSU(..., "ANDREAS", ...)
CALL CMSCSP(..., "THASOS", ...)
5 CALL CMALLC(...)
...
```

CPI-C side information

	Symbolic destination name	Partner LU name	Partner TP name	Mode name	Security type	User ID	Password
	VENUS	GALAXY.SOLARLU	VENUSTP	#INTER	CM_SECURITY_NONE		
1	MARS	GALAXY.SOLARLU	MARSTP	#INTER	CM_SECURITY_NONE		
	SATURN	GALAXY.SOLARLU	SATURNTP	#INTER	CM_SECURITY_NONE		
	PLUTO	GALAXY.SOLARLU	PLUTOTP	#INTER	CM_SECURITY_NONE		

Logical unit
Characteristics of the conversation
6
Attach request (FMH-5) to partner LU
ANDREAS, THASOS
4
Set:
• Security type
• User ID
• Password
2
Initialize the characteristics of the conversation

Figure 66. IBM MQ support for conversation level authentication

On IBM i, UNIX systems, and Windows systems, an MCA uses Common Programming Interface Communications (CPI-C) calls to communicate with a partner MCA across an SNA network. In the channel definition at the caller end of a channel, the value of the CONNAME parameter is a symbolic destination name, which identifies a CPI-C side information entry (1). This entry specifies:

- The name of the partner LU
- The name of the partner TP, which is a responder MCA
- The name of the mode to be used for the conversation

A side information entry can also specify the following security information:

- A security type.

The commonly implemented security types are CM_SECURITY_NONE, CM_SECURITY_PROGRAM, and CM_SECURITY_SAME, but others are defined in the CPI-C specification.
- A user ID.
- A password.

A caller MCA prepares to allocate a conversation with a responder MCA by issuing the CPI-C call CMINIT, using the value of CONNAME as one of the parameters on the call. The CMINIT call identifies, for the benefit of the local LU, the side information entry that the MCA intends to use for the conversation. The local LU uses the values in this entry to initialize the characteristics of the conversation (2).

The caller MCA then checks the values of the USERID and PASSWORD parameters in the channel definition (3). If USERID is set, the caller MCA issues the following CPI-C calls (4):
- CMSCST, to set the security type for the conversation to CM_SECURITY_PROGRAM.
- CMSCSU, to set the user ID for the conversation to the value of USERID.
- CMSCSP, to set the password for the conversation to the value of PASSWORD. CMSCSP is not called unless PASSWORD is set.

The security type, user ID, and password set by these calls override any values acquired previously from the side information entry.

The caller MCA then issues the CPI-C call CMALLC to allocate the conversation (5). In response to this call, the local LU sends an attach request (Function Management Header 5, or FMH-5) to the partner LU (6).

If the partner LU will accept a user ID and a password, the values of USERID and PASSWORD are included in the attach request. If the partner LU will not accept a user ID and a password, the values are not included in the attach request. The local LU discovers whether the partner LU will accept a user ID and a password as part of an exchange of information when the LUs bind to form a session.

In a later version of the attach request, a password substitute can flow between the LUs instead of a clear password. A password substitute is a DES Message Authentication Code (MAC), or an SHA-1 message digest, formed from the password. Password substitutes can be used only if both LUs support them.

When the partner LU receives an incoming attach request containing a user ID and a password, it might use the user ID and password for the purposes of identification and authentication. By referring to access control lists, the partner LU might also determine whether the user ID has the authority to allocate a conversation and attach the responder MCA.

In addition, the responder MCA might run under the user ID included in the attach request. In this case, the user ID becomes the default user ID for the responder MCA and is used for authority checks when the MCA attempts to connect to the queue manager. It might also be used for authority checks subsequently when the MCA attempts to access the queue manager's resources.

The way in which a user ID and a password in an attach request can be used for identification, authentication, and access control is implementation dependent. For information specific to your SNA subsystem, refer to the appropriate documentation.

If USERID is not set, the caller MCA does not call CMSCST, CMSCSU, and CMSCSP. In this case, the security information that flows in an attach request is determined solely by what is specified in the side information entry and what the partner LU will accept.

Conversation level authentication and IBM MQ for z/OS

Use this topic to gain an overview of how conversation level authentication works, on z/OS.

On IBM MQ for z/OS, MCAs do not use CPI-C. Instead, they use APPC/MVS TP Conversation Callable Services, an implementation of Advanced Program-to-Program Communication (APPC), which has some CPI-C features. When a caller MCA allocates a conversation, a security type of SAME is specified on the call. Therefore, because an APPC/MVS LU supports persistent verification only for inbound conversations, not for outbound conversations, there are two possibilities:

- If the partner LU trusts the APPC/MVS LU and will accept an already verified user ID, the APPC/MVS LU sends an attach request containing:
  - The channel initiator address space user ID
  - A security profile name, which, if RACF is used, is the name of the current connect group of the channel initiator address space user ID
  - An already verified indicator
- If the partner LU does not trust the APPC/MVS LU and will not accept an already verified user ID, the APPC/MVS LU sends an attach request containing no security information.

On IBM MQ for z/OS, the USERID and PASSWORD parameters on the DEFINE CHANNEL command cannot be used for a message channel and are valid only at the client connection end of an MQI channel. Therefore, an attach request from an APPC/MVS LU never contains values specified by these parameters.

Security for queue manager clusters

Though queue manager clusters can be convenient to use, you must pay special attention to their security.

A *queue manager cluster* is a network of queue managers that are logically associated in some way. A queue manager that is a member of a cluster is called a *cluster queue manager*.

A queue that belongs to a cluster queue manager can be made known to other queue managers in the cluster. Such a queue is called a *cluster queue*. Any queue manager in a cluster can send messages to cluster queues without needing any of the following:

- An explicit remote queue definition for each cluster queue
- Explicitly defined channels to and from each remote queue manager
- A separate transmission queue for each outbound channel

You can create a cluster in which two or more queue managers are clones. This means that they have instances of the same local queues, including any local queues declared as cluster queues, and can support instances of the same server applications.

When an application connected to a cluster queue manager sends a message to a cluster queue that has an instance on each of the cloned queue managers, IBM MQ decides which queue manager to send it to. When many applications send messages to the cluster queue, IBM MQ balances the workload across each of the queue managers that have an instance of the queue. If one of the systems hosting a cloned queue manager fails, IBM MQ continues to balance the workload across the remaining queue managers until the system that failed is restarted.

If you are using queue manager clusters, you need to consider the following security issues:

- Allowing only selected queue managers to send messages to your queue manager
- Allowing only selected users of a remote queue manager to send messages to a queue on your queue manager
- Allowing applications connected to your queue manager to send messages only to selected remote queues

These considerations are relevant even if you are not using clusters, but they become more important if you are using clusters.

If an application can send messages to one cluster queue, it can send messages to any other cluster queue without needing additional remote queue definitions, transmission queues, or channels. It therefore becomes more important to consider whether you need to restrict access to the cluster queues on your queue manager, and to restrict the cluster queues to which your applications can send messages.

There are some additional security considerations, which are relevant only if you are using queue manager clusters:

- Allowing only selected queue managers to join a cluster
- Forcing unwanted queue managers to leave a cluster

For more information about all these considerations, see Keeping clusters secure. z/OS For considerations specific to IBM MQ for z/OS, see "Security in queue manager clusters on z/OS" on page 643.

Related tasks:
"Preventing queue managers receiving messages" on page 817
You can prevent a cluster queue manager from receiving messages it is unauthorized to receive by using exit programs.

Security for IBM MQ Publish/Subscribe

There are additional security considerations if you are using IBM MQ Publish/Subscribe.

In a publish/subscribe system, there are two types of application: publisher and subscriber. *Publishers* supply information in the form of IBM MQ messages. When a publisher publishes a message, it specifies a *topic*, which identifies the subject of the information inside the message.

Subscribers are the consumers of the information that is published. A subscriber specifies the topics it is interested in by subscribing to them.

The *queue manager* is an application supplied with IBM MQ Publish/Subscribe. It receives published messages from publishers and subscription requests from subscribers, and routes the published messages to the subscribers. A subscriber is sent messages only on those topics to which it has subscribed.

For more information, see Publish/subscribe security.

Multicast security

Use this information to understand why security processes might be needed with IBM MQ Multicast.

IBM MQ Multicast does not have in-built security. Security checks are handled in the queue manager at MQOPEN time and the MQMD field setting is handled by the client. Some applications in the network might not be IBM MQ applications (For example, LLM applications, see Multicast interoperability with IBM MQ Low Latency Messaging for more information), therefore you might need to implement your own security procedures because receiving applications cannot be certain of the validity of context fields.

There are three security processes to consider:

Access control

Access control in IBM MQ is based on user IDs. For more information on this subject, see "Access control for clients" on page 467.

Network security

An isolated network might be a viable security option to prevent fake messages. It is possible for an application on the multicast group address to publish malicious messages using native communication functions, which are indistinguishable from MQ messages because they come from an application on the same multicast group address.

It is also possible for a client on the multicast group address to receive messages that were intended for other clients on the same multicast group address.

Isolating the multicast network ensures that only valid clients and applications have access. This security precaution can prevent malicious messages from coming in, and confidential information from going out.

For information about multicast group network addresses, see: Setting the appropriate network for multicast traffic

Digital signatures

A digital signature is formed by encrypting a representation of a message. The encryption uses the private key of the signatory and, for efficiency, usually operates on a message digest rather than the message itself. Digitally signing a message before an MQPUT is a good security precaution, but this process might have a detrimental effect on performance if there is a large volume of messages.

Digital signatures vary with the data being signed. If two different messages are signed digitally by the same entity, the two signatures differ, but both signatures can be verified with the same public key, that is, the public key of the entity that signed the messages.

As mentioned previously in this section, it might be possible for an application on the multicast group address to publish malicious messages using native communication functions, which are indistinguishable from MQ messages. Digital signatures provide proof of origin, and only the sender knows the private key, which provides strong evidence that the sender is the originator of the message.

For more information on this subject, see "Cryptographic concepts" on page 380.

Firewalls and Internet pass-thru

You would normally use a firewall to prevent access from hostile IP addresses, for example in a Denial of Service attack. However, you might need to temporarily block IP addresses within IBM MQ, perhaps while you wait for a security administrator to update the firewall rules.

To block one or more IP addresses, create a channel authentication record of type BLOCKADDR or ADDRESSMAP. For more information, see “Blocking specific IP addresses” on page 762.

Security for IBM MQ internet pass-thru

Internet pass-thru can simplify communication through a firewall, but this has security implications.

IBM MQ internet pass-thru is an IBM MQ base product extension that is supplied in SupportPac MS81.

IBM MQ internet pass-thru enables two queue managers to exchange messages, or an IBM MQ client application to connect to a queue manager, over the Internet without requiring a direct TCP/IP connection. This is useful if a firewall prohibits a direct TCP/IP connection between two systems. It makes the passage of IBM MQ channel protocol flows into and out of a firewall simpler and more manageable by tunnelling the flows inside HTTP or by acting as a proxy. Using the Secure Sockets Layer (SSL), it can also be used to encrypt and decrypt messages that are sent over the Internet.

When your IBM MQ system communicates with IPT, unless you are using SSLProxyMode in IPT, ensure that the CipherSpec used by IBM MQ matches the CipherSuite used by IPT:

- When IPT is acting as the SSL or TLS server and IBM MQ is connecting as the SSL or TLS client, the CipherSpec used by IBM MQ must correspond to a CipherSuite that is enabled in the relevant IPT key ring.
- When IPT is acting as the SSL or TLS client and is connecting to an IBM MQ SSL or TLS server, the IPT CipherSuite must match the CipherSpec defined on the receiving IBM MQ channel.

If you migrate from IPT to the integrated IBM MQ SSL and TLS support, transfer the digital certificates from IPT Using iKeyman.

For more information about IBM MQ internet pass-thru, see *MS81: IBM MQ internet pass-thru*, available from the following address: http://www.ibm.com/software/integration/support/supportpacs/

IBM MQ for z/OS security implementation checklist

This topic gives a step-by-step procedure you can use to work out and define the security implementation for each of your IBM MQ queue managers.

RACF provides definitions for the IBM MQ security classes in its supplied static Class Descriptor Table (CDT). As you work through the checklist, you can determine which of these classes your setup requires. You must ensure that they are activated as described in "RACF security classes" on page 567.

Refer to other sections for details, in particular "Profiles used to control access to IBM MQ resources" on page 577.

If you require security checking, follow this checklist to implement it:

1. Activate the RACF MQADMIN (uppercase profiles) or MXADMIN (mixed case profiles) class.
   - Do you want security at queue-sharing group level, queue-manager level, or a combination of both?

     See, "Profiles to control queue-sharing group or queue manager level security" on page 572.
2. Do you need connection security?
   - **Yes**: Activate the MQCONN class. Define appropriate connection profiles at either queue manager level or queue-sharing group level in the MQCONN class. Then permit the appropriate users or groups access to these profiles.

     Note: Only users of the **MQCONN** API request or CICS or IMS address space user IDs need to have access to the corresponding connection profile.
   - **No**: Define an hlq.NO.CONNECT.CHECKS profile at either queue manager level or queue-sharing group level in the MQADMIN or MXADMIN class.
3. Do you need security checking on commands?
   - **Yes**: Activate the MQCMDS class. Define appropriate command profiles at either queue manager level or queue-sharing group level in the MQCMDS class. Then permit the appropriate users or groups access to these profiles.

     If you are using a queue-sharing group, you might need to include the user IDs used by the queue manager itself and the channel initiator. See "Setting up IBM MQ for z/OS resource security" on page 634.
   - **No**: Define an hlq.NO.CMD.CHECKS profile for the required queue manager or queue-sharing group in the MQADMIN or MXADMIN class.
4. Do you need security on the resources used in commands?
   - **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define appropriate profiles for protecting resources on commands at either queue manager level or queue-sharing group level in the MQADMIN or MXADMIN class. Then permit the appropriate users or groups access to these profiles. Set the CMDUSER parameter in CSQ6SYSP to the default user ID to be used for command security checks.

     If you are using a queue-sharing group, you might need to include the user IDs used by the queue manager itself and the channel initiator. See "Setting up IBM MQ for z/OS resource security" on page 634.
   - **No**: Define an hlq.NO.CMD.RESC.CHECKS profile for the required queue manager or queue-sharing group in the MQADMIN or MXADMIN class.
5. Do you need queue security?

- **Yes**: Activate the MQQUEUE or MXQUEUE class. Define appropriate queue profiles for the required queue manager or queue-sharing group in the MQQUEUE or MXQUEUEclass. Then permit the appropriate users or groups access to these profiles.
- **No**: Define an hlq.NO.QUEUE.CHECKS profile for the required queue manager or queue-sharing group in the MQADMIN or MXADMIN class.

6. Do you need process security?
   - **Yes**: Activate the MQPROC or MXPROC class. Define appropriate process profiles at either queue manager or queue-sharing group level and permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.PROCESS.CHECKS profile for the appropriate queue manager or queue-sharing group in the MQADMIN or MXADMIN class.
7. Do you need namelist security?
   - **Yes**: Activate the MQNLIST or MXNLISTclass. Define appropriate namelist profiles at either queue manager level or queue-sharing group level in the MQNLIST or MXNLIST class. Then permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.NLIST.CHECKS profile for the required queue manager or queue-sharing group in the MQADMIN or MXADMIN class.
8. Do you need topic security?
   - **Yes**: Activate the MXTOPIC class. Define appropriate topic profiles at either queue manager level or queue-sharing group level in the MXTOPIC class. Then permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.TOPIC.CHECKS profile for the required queue manager or queue-sharing group in the MQADMIN or MXADMIN class.
9. Do any users need to protect the use of the **MQOPEN** or **MQPUT1** options relating to the use of context?
   - **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define hlq.CONTEXT.queuename profiles at the queue, queue manager, or queue-sharing group level in the MQADMIN or MXADMIN class. Then permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.CONTEXT.CHECKS profile for the required queue manager or queue-sharing group in the MQADMIN or MXADMIN class.
10. Do you need to protect the use of alternative user IDs?
    - **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define the appropriate hlq.ALTERNATE.USER. *alternateuserid* profiles for the required queue manager or queue-sharing group and permit the required users or groups access to these profiles.
    - **No**: Define the profile hlq.NO.ALTERNATE.USER.CHECKS for the required queue manager or queue-sharing group in the MQADMIN or MXADMIN class.
11. Do you need to tailor which user IDs are to be used for resource security checks through RESLEVEL?
    - **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define an hlq.RESLEVEL profile at either queue manager level or queue-sharing group level in the MQADMIN or MXADMIN class. Then permit the required users or groups access to the profile.
    - **No**: Ensure that no generic profiles exist in the MQADMIN or MXADMIN class that can apply to hlq.RESLEVEL. Define an hlq.RESLEVEL profile for the required queue manager or queue-sharing group and ensure that no users or groups have access to it.
12. Do you need to 'timeout' unused user IDs from IBM MQ ?
    - **Yes**: Determine what timeout values you would like to use and issue the MQSC ALTER SECURITY command to change the TIMEOUT and INTERVAL parameters.
    - **No**: Issue the MQSC ALTER SECURITY command to set the INTERVAL value to zero.

Note: Update the CSQINP1 initialization input data set used by your subsystem so that the MQSC ALTER SECURITY command is issued automatically when the queue manager is started.

13. Do you use distributed queuing?
    - **Yes**: Use channel authentication records. For more information, see "Channel authentication records" on page 424.
    - You can also determine the appropriate MCAUSER attribute value for each channel, or provide suitable channel security exits.
14. Do you want to use the Secure Sockets Layer (SSL)?
    - **Yes**: To specify that any user presenting an SSL/TLS personal certificate containing a specified DN is to use a specific MCAUSER, set a channel authentication record of type SSLPEERMAP. You can specify a single distinguished name or a pattern including wildcards.
    - Plan your SSL infrastructure. Install the System SSL feature of z/OS. In RACF, set up your certificate name filters (CNFs), if you are using them, and your digital certificates. Set up your SSL key ring. Ensure that the SSLKEYR queue manager attribute is nonblank and points to your SSL key ring. Also ensure that the value of SSLTASKS is at least 2.
    - **No**: Ensure that SSLKEYR is blank, and SSLTASKS is zero.

    For further details about SSL, see "SSL and TLS security protocols in IBM MQ" on page 397.
15. Do you use clients?
    - **Yes**: Use channel authentication records.
    - You can also determine the appropriate MCAUSER attribute value for each server-connection channel, or provide suitable channel security exits if required.
16. Check your switch settings.

    IBM MQ issues messages when the queue manager is started that display your security settings. Use these messages to determine whether your switches are set correctly.
17. Do you send passwords from client applications?
    - **Yes**: Ensure that the z/OS feature is installed and Integrated Cryptographic Service Facility (ICSF) is started for the best protection.
    - **No**: You can ignore the error message reporting that ICSF has not started.

    For further information about ICSF see "Using the Integrated Cryptographic Service Facility (ICSF)" on page 643

Setting up security

This collection of topics contains information specific to different operating systems, and to the use of clients.

Setting up security on Windows, UNIX and Linux systems

Security considerations specific to Windows, UNIX and Linux systems.

IBM MQ queue managers transfer information that is potentially valuable, so you need to use an authority system to ensure that unauthorized users cannot access your queue managers. Consider the following types of security controls:

Who can administer IBM MQ
: You can define the set of users who can issue commands to administer IBM MQ.

Who can use IBM MQ objects
: You can define which users (usually applications) can use MQI calls and PCF commands to do the following:
- Who can connect to a queue manager.
- Who can access objects (queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects), and what type of access they have to those objects.
- Who can access IBM MQ messages.
- Who can access the context information associated with a message.

Channel security
: You need to ensure that channels used to send messages to remote systems can access the required resources.

You can use standard operating facilities to grant access to program libraries, MQI link libraries, and commands. However, the directory containing queues and other queue manager data is private to IBM MQ; do not use standard operating system commands to grant or revoke authorizations to MQI resources.

Connecting to IBM MQ using Terminal Services

The **`Create global objects`** user right can cause problems if you are using Terminal Services.

If you are connecting to a Windows system by using Terminal Services and you have problems creating or starting a queue manager, this might be because of the user right, **`Create global objects`**, in recent versions of Windows.

The **`Create global objects`** user right limits the users authorized to create objects in the global namespace. In order for an application to create a global object, it must either be running in the global namespace, or the user under which the application is running must have the **`Create global objects`** user right applied to it.

Administrators have the **`Create global objects`** user right applied by default, so an administrator can create and start queue managers when connected by using Terminal Services without altering the user rights.

If the various methods of administering IBM MQ do no work when you use terminal services, try setting the **`Create global objects`** user right:

1. Open the Administrative Tools panel:

   Windows Server 2008 and Windows Server 2012
   : Access this panel using **Control Panel** > **System and Maintenance** > **Administrative Tools**.

   Windows 7 and Windows 8.1
   : Access this panel using **Administrative Tools** > **Computer Management**
2. Double-click **Local Security Policy**.
3. Expand **Local Policies** .
4. Click **User Rights Assignment** .
5. Add the new user or group to the `Create global objects` policy.

Creating and managing groups on Windows

These instructions lead you through the process of administering groups on a workstation or member server machine.

For domain controllers, users and groups are administered through Active Directory. For more details on using Active Directory refer to the appropriate operating system instructions.

Any changes you make to a principal's group membership are not recognized until the queue manager is restarted, or you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

Use the Computer Management panel to work with user and groups. Any changes made to the current logged on user might not be effective until the user logs in again.

Windows Server 2008 and Windows Server 2012
: Access this panel using **Control Panel** > **System and Maintenance** > **Administrative Tools** > **Computer Management**.

Windows 7 and Windows 8.1
: Access this panel using **Administrative Tools** > **Computer Management**

Creating a group on Windows

Create a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**. The Administrative Tools panel opens.
3. Double-click **Computer Management**. The Computer Management panel opens.
4. Expand **Local Users and Groups**.
5. Right-click **Groups**, and select **New Group...**. The New Group panel is displayed.
6. Type an appropriate name in the Group name field, then click **Create**.
7. Click **Close**.

Adding a user to a group on Windows

Add a user to a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**. The Administrative Tools panel opens.
3. Double-click **Computer Management**. The Computer Management panel opens.
4. From the Computer Management panel, expand **Local Users and Groups**.
5. Select **Users**
6. Double-click the user that you want to add to a group. The user properties panel is displayed.
7. Select the **Member Of** tab.
8. Select the group that you want to add the user to. If the group you want is not visible:
   a. Click **Add...**. The Select Groups panel is displayed.
   b. Click **Locations...**. The Locations panel is displayed.
   c. Select the location of the group you want to add the user to from the list and click **OK**.
   d. Type the group name in the field provided.

      Alternatively, click **Advanced...** and then **Find Now** to list the groups available in the currently selected location. From here, select the group you want to add the user to and click **OK**.
   e. Click **OK**. The user properties panel is displayed, showing the group you added.
   f. Select the group.
9. Click **OK**. The Computer Management panel is displayed.

Displaying who is in a group on Windows

Display the members of a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**. The Administrative Tools panel opens.
3. Double-click **Computer Management**. The Computer Management panel opens.
4. From the Computer Management panel, expand **Local Users and Groups**.
5. Select **Groups**.
6. Double-click a group. The group properties panel is displayed. The group properties panel is displayed.

Results

The group members are displayed.

Removing a user from a group on Windows

Remove a user from a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**. The Administrative Tools panel opens.
3. Double-click **Computer Management**. The Computer Management panel opens.
4. From the Computer Management panel, expand **Local Users and Groups**.
5. Select **Users**.
6. Double-click the user that you want to add to a group. The user properties panel is displayed.
7. Select the **Member Of** tab.
8. Select the group that you want to remove the user from, then click **Remove**.
9. Click **OK**. The Computer Management panel is displayed.

Results

You have now removed the user from the group.

Creating and managing groups on HP-UX

On HP-UX, providing you are not using NIS or NIS+, use the System Administration Manager (SAM) to work with groups.

Creating a group on HP-UX

Add a user to a group by using the System Administration Manager

Procedure

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Select Add from the Actions pull down to display the Add a New Group panel.
4. Enter the name of the group and select the users that you want to add to the group.
5. Click Apply to create the group.

Results

You have now created a group.

Adding a user to a group on HP-UX

Add a user to a group by using the System Administration Manager.

Procedure

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel.
4. Select a user that you want to add to the group and click Add.
5. If you want to add other users to the group, repeat step 4 for each user.
6. When you have finished adding names to the list, click OK.

Results

You have now added a user to a group.

Displaying who is in a group on HP-UX

Display who is in a group by using the System Administration Manager

Procedure

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel, showing a list of the users in the group.

Results

The group members are displayed.

Removing a user from a group on HP-UX

Remove a user from a group by using the System Administration Manager.

Procedure

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel.
4. Select a user that you want to remove from the group and click Remove.
5. If you want to remove other users from the group, repeat step 4 for each user.
6. When you have finished removing names from the list, click OK.

Results

You have now removed a user from a group

Creating and managing groups on AIX

On AIX, providing you are not using NIS or NIS+, use SMITTY to work with groups.

Creating a group

Create a group using SMITTY.

Procedure

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Add a Group and press Enter.
4. Enter the name of the group and the names of any users that you want to add to the group, separated by commas.
5. Press Enter to create the group.

Results

You have now created a group.

Adding a user to a group

Add a user to a group by using SMITTY.

Procedure

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.
5. Add the names of the users that you want to add to the group, separated by commas.
6. Press Enter to add the names to the group.

Displaying who is in a group

Display who is in a group using SMITTY.

Procedure

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.

Results

The group members are displayed.

Removing a user from a group

Remove a user from a group by using SMITTY.

Procedure

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.
5. Delete the names of the users that you want to remove from the group.
6. Press Enter to remove the names from the group.

Results

You have now removed a user from a group.

Creating and managing groups on Solaris

On Solaris, providing you are not using NIS or NIS+, use the `/etc/group` file to work with groups.

Creating a group on Solaris

Creating a group by using the **`groupadd`** command.

Procedure

Type the following command: `groupadd` *`group-name`* where *group-name* is the name of the group.

Results

The file `/etc/group` file holds group information.

Adding a user to a group on Solaris

Add a user to a group by using the **`usermod`** command.

Procedure

To add a member to a supplementary group, execute the `usermod` command and list the supplementary groups that the user is currently a member of, and the supplementary groups that the user is to become a member of. For example, if the user is a member of the group `groupa`, and is to become a member of `groupb` also, use the following command: `usermod -G groupa,groupb` *`user-name`*, where *user-name* is the user name.

Displaying who is in a group on Solaris

To discover who is a member of a group, look at the entry for that group in the `/etc/group` file.

Removing a user from a group on Solaris

Remove a user from a group by using the **`usermod`** command.

Procedure

To remove a member from a supplementary group, execute the **`usermod`** command listing the supplementary groups that you want the user to remain a member of. For example, if the user's primary group is `users` and the user is also a member of the groups `mqm`, `groupa` and `groupb`, to remove the user from the `mqm` group, the following command is used: `usermod -G groupa,groupb` *`user-name`*, where *`user-name`* is the user name.

Creating and managing groups on Linux

On Linux, providing you are not using NIS or NIS+, use the `/etc/group` file to work with groups.

Creating a group on Linux

Create a group by using the **groupadd** command.

Procedure

To create a new group, type the following command: `groupadd -g` *group-ID group-name* , where *group-ID* is the numeric identifier of the group, and *group-name* is the name of the group.

Results

The file `/etc/group` file holds group information.

Adding a user to a group on Linux

Add a user to a group by using the **usermod** command.

Procedure

To add a member to a supplementary group, execute the `usermod` command and list the supplementary groups that the user is currently a member of, and the supplementary groups that the user is to become a member of. For example, if the user is a member of the group `groupa`, and is to become a member of `groupb` also, the following command is used: `usermod -G groupa,groupb` *user-name* , where *user-name* is the user name.

Displaying who is in a group on Linux

Display who is in a group by using the **getent** command.

Procedure

To display who is a member of a group, type the following command: `getent group` *group-name* , where *group-name* is the name of the group.

Removing a user from a group

Remove a user from a group by using the **usermod** command.

Procedure

To remove a member from a supplementary group, execute the **usermod** command listing the supplementary groups that you want the user to remain a member of. For example, if the user's primary group is `users` and the user is also a member of the groups `mqm`, `groupa` and `groupb`, to remove the user from the `mqm` group, the following command is used: `usermod -G groupa,groupb` *user-name* , where *user-name* is the user name.

How authorizations work

The authorization specification tables in the topics in this section define precisely how the authorizations work and the restrictions that apply.

The tables apply to these situations:

- Applications that issue MQI calls
- Administration programs that issue MQSC commands as escape PCFs
- Administration programs that issue PCF commands

In this section, the information is presented as a set of tables that specify the following:

Action to be performed
MQI option, MQSC command, or PCF command.

Access control object
Queue, process, queue manager, namelist, authentication information, channel, client connection channel, listener, or service.

Authorization required
Expressed as an MQZAO_ constant.

In the tables, the constants prefixed by MQZAO_ correspond to the keywords in the authorization list for the **setmqaut** command for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword `+browse`, MQZAO_SET_ALL_CONTEXT corresponds to the keyword `+setall`, and so on. These constants are defined in the header file cmqzc.h, supplied with the product.

Authorizations for MQI calls

MQCONN, **MQOPEN**, **MQPUT1**, and **MQCLOSE** might require authorization checks. The tables in this topic summarize the authorizations needed for each call.

An application is allowed to issue specific MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls might require authorization checks: **MQCONN**, **MQOPEN**, **MQPUT1**, and **MQCLOSE**.

For **MQOPEN** and **MQPUT1** , the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application might be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of resolving a name that is not a queue manager alias, unless the queue manager alias definition is opened directly; that is, its name is displayed in the *ObjectName* field of the object descriptor. Authority is always needed for the object being opened. In some cases additional queue-independent authority, obtained through an authorization for the queue manager object, is required.

Table 23 on page 520, Table 24 on page 520, Table 25 on page 521, and Table 26 on page 521 summarize the authorizations needed for each call. In the tables *Not applicable* means that authorization checking is not relevant to this operation; *No check* means that no authorization checking is performed.

Note: You will find no mention of namelists, channels, client connection channels, listeners, services, or authentication information objects in these tables. This is because none of the authorizations apply to these objects, except for MQOO_INQUIRE, for which the same authorizations apply as for the other objects.

The special authorization MQZAO_ALL_MQI includes all the authorizations in the tables that are relevant to the object type, except MQZAO_DELETE and MQZAO_DISPLAY, which are classed as administration authorizations.

In order to modify any of the message context options, you must have the appropriate authorizations to issue the call. For example, in order to use MQOO_SET_IDENTITY_CONTEXT or MQPMO_SET_IDENTITY_CONTEXT, you must have `+setid` permission.

Table 23. Security authorization needed for MQCONN calls

Authorization required for:	**Queue object (1 on page 521)**	**Process object**	**Queue manager object**
MQCONN	Not applicable	Not applicable	MQZAO_CONNECT

Table 24. Security authorization needed for MQOPEN calls

Authorization required for:	**Queue object (1 on page 521)**	**Process object**	**Queue manager object**
MQOO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT **(2 on page 521)**	MQZAO_INPUT	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) **(3 on page 521)**	MQZAO_OUTPUT	Not applicable	Not applicable
MQOO_PASS_ IDENTITY_CONTEXT **(4 on page 521)**	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ALL_ CONTEXT **(4 on page 521, 5 on page 521)**	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT **(4 on page 521, 5 on page 521)**	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(6 on page 521)**
MQOO_SET_ ALL_CONTEXT **(4 on page 521, 7 on page 521)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6 on page 521)**
MQOO_OUTPUT (Transmission queue) **(8 on page 521)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6 on page 521)**
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	**(9 on page 521)**	**(9 on page 521)**	MQZAO_ALTERNATE_ USER_AUTHORITY **(9 on page 521, 10 on page 522)**

Table 25. Security authorization needed for MQPUT1 calls

Authorization required for:	**Queue object (1)**	**Process object**	**Queue manager object**
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_CONTEXT **(11 on page 522)**	Not applicable	No check
MQPMO_PASS_ALL _CONTEXT	MQZAO_PASS_ ALL_CONTEXT **(11 on page 522)**	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_CONTEXT **(11 on page 522)**	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(6)**
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT **(11 on page 522)**	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6)**
(Transmission queue) **(8)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6)**
MQPMO_ALTERNATE_ USER_AUTHORITY	**(12 on page 522)**	Not applicable	MQZAO_ALTERNATE_ USER_AUTHORITY **(10 on page 522)**

Table 26. Security authorization needed for MQCLOSE calls

Authorization required for:	**Queue object (1)**	**Process object**	**Queue manager object**
MQCO_DELETE	MQZAO_DELETE **(13 on page 522)**	Not applicable	Not applicable
MQCO_DELETE _PURGE	MQZAO_DELETE **(13 on page 522)**	Not applicable	Not applicable

Notes for the tables:

1. If opening a model queue:
   - MQZAO_DISPLAY authority is needed for the model queue, in addition to the authority to open the model queue for the type of access for which you are opening.
   - MQZAO_CREATE authority is not needed to create the dynamic queue.
   - The user identifier used to open the model queue is automatically granted all the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
3. This check is performed for all output cases, except transmission queues (see note 8).
4. MQOO_OUTPUT must also be specified.
5. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
6. This authority is required for both the queue manager object and the particular queue.
7. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
8. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
9. At least one of MQOO_INQUIRE (for any object type), or MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET (for queues) must also be specified. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.

10. This authorization allows any *AlternateUserId* to be specified.
11. An MQZAO_OUTPUT check is also carried out if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
12. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
13. The check is carried out only if both of the following are true:
    - A permanent dynamic queue is being closed and deleted.
    - The queue was not created by the **MQOPEN** call that returned the object handle being used.

    Otherwise, there is no check.

Authorizations for MQSC commands in escape PCFs

This information summarizes the authorizations needed for each MQSC command contained in Escape PCF.

Not applicable means that this operation is not relevant to this object type.

The user ID under which the program that submits the command is running must also have the following authorities:
- MQZAO_CONNECT authority to the queue manager
- MQZAO_DISPLAY authority on the queue manager in order to perform PCF commands
- Authority to issue the MQSC command within the text of the Escape PCF command

`ALTER object`

Object	**Authorization required**
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

`CLEAR object`

Object	Authorization required
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable
Communication information	Not applicable

DEFINE *object* NOREPLACE (1 on page 526)

Object	Authorization required
Queue	MQZAO_CREATE **(2 on page 526)**
Topic	MQZAO_CREATE **(2 on page 526)**
Process	MQZAO_CREATE **(2 on page 526)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(2 on page 526)**
Authentication information	MQZAO_CREATE **(2 on page 526)**
Channel	MQZAO_CREATE **(2 on page 526)**
Client connection channel	MQZAO_CREATE **(2 on page 526)**
Listener	MQZAO_CREATE **(2 on page 526)**
Service	MQZAO_CREATE **(2 on page 526)**
Communication information	MQZAO_CREATE **(2 on page 526)**

DEFINE *object* REPLACE (1 on page 526, 3 on page 526)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

DELETE *object*

Object	Authorization required
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	Not applicable
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE
Communication information	MQZAO_DELETE

DISPLAY *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY
Communication information	MQZAO_DISPLAY

START *object*

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

STOP *object*

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

Channel Commands

Command	Object	Authorization required
PING CHANNEL	Channel	MQZAO_CONTROL
RESET CHANNEL	Channel	MQZAO_CONTROL_EXTENDED
RESOLVE CHANNEL	Channel	MQZAO_CONTROL_EXTENDED

Subscription Commands

Command	Object	Authorization required
ALTER SUB	Topic	MQZAO_CONTROL
DEFINE SUB	Topic	MQZAO_CONTROL
DELETE SUB	Topic	MQZAO_CONTROL
DISPLAY SUB	Topic	MQZAO_DISPLAY

Security Commands

Command	Object	Authorization required
SET AUTHREC	Queue manager	MQZAO_CHANGE
DELETE AUTHREC	Queue manager	MQZAO_CHANGE
DISPLAY AUTHREC	Queue manager	MQZAO_DISPLAY
DISPLAY AUTHSERV	Queue manager	MQZAO_DISPLAY
DISPLAY ENTAUTH	Queue manager	MQZAO_DISPLAY
SET CHLAUTH	Queue manager	MQZAO_CHANGE
DISPLAY CHLAUTH	Queue manager	MQZAO_DISPLAY
REFRESH SECURITY	Queue manager	MQZAO_CHANGE

Status Displays

Command	Object	Authorization required
DISPLAY CHSTATUS	Queue manager	MQZAO_DISPLAY Note that `+inq` authority (or equivalently MQZAO_INQUIRE) is required on the transmission queue if the channel type is CLUSSDR.
DISPLAY LSSTATUS	Queue manager	MQZAO_DISPLAY
DISPLAY PUBSUB	Queue manager	MQZAO_DISPLAY
DISPLAY SBSTATUS	Queue manager	MQZAO_DISPLAY
DISPLAY SVSTATUS	Queue manager	MQZAO_DISPLAY
DISPLAY TPSTATUS	Queue manager	MQZAO_DISPLAY

Cluster Commands

Command	Object	Authorization required
DISPLAY CLUSQMGR	Queue manager	MQZAO_DISPLAY
REFRESH CLUSTER	'mqm' group membership required	
RESET CLUSTER	'mqm' group membership required	
SUSPEND QMGR	'mqm' group membership required	
RESUME QMGR	'mqm' group membership required	

Other Administrative Commands

Command	Object	Authorization required
PING QMGR	Queue manager	MQZAO_DISPLAY
REFRESH QMGR	Queue manager	MQZAO_CHANGE
RESET QMGR	Queue manager	MQZAO_CHANGE
DISPLAY CONN	Queue manager	MQZAO_DISPLAY
STOP CONN	Queue manager	MQZAO_CHANGE

Note:

1. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the **setmqaut** command.
3. This applies if the object to be replaced already exists. If it does not, the check is as for DEFINE *object* NOREPLACE.

Related information:

Clustering: Using REFRESH CLUSTER best practices

Authorizations for PCF commands

This section summarizes the authorizations needed for each PCF command.

No check means that no authorization checking is carried out; *Not applicable* means that this operation is not relevant to this object type.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- MQZAO_DISPLAY authority on the queue manager in order to perform PCF commands

The special authorization MQZAO_ALL_ADMIN includes all the authorizations in the following list that are relevant to the object type, except MQZAO_CREATE, which is not specific to a particular object or object type.

Change *object*

Object	**Authorization required**
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

Clear *object*

Object	**Authorization required**
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable
Communication information	Not applicable

Copy *object* (without replace) (1)

Object	Authorization required
Queue	MQZAO_CREATE **(2)**
Topic	MQZAO_CREATE **(2)**
Process	MQZAO_CREATE **(2)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(2)**
Authentication information	MQZAO_CREATE **(2)**
Channel	MQZAO_CREATE **(2)**
Client connection channel	MQZAO_CREATE **(2)**
Listener	MQZAO_CREATE **(2)**
Service	MQZAO_CREATE **(2)**
Communication information	MQZAO_CREATE **(2 on page 532)**

Copy *object* (with replace) (1, 4)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

Create *object* (without replace) (3)

Object	Authorization required
Queue	MQZAO_CREATE **(2)**
Topic	MQZAO_CREATE **(2)**
Process	MQZAO_CREATE **(2)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(2)**
Authentication information	MQZAO_CREATE **(2)**
Channel	MQZAO_CREATE **(2)**
Client connection channel	MQZAO_CREATE **(2)**
Listener	MQZAO_CREATE **(2)**
Service	MQZAO_CREATE **(2)**
Communication information	MQZAO_CREATE **(2)**

Create *object* **(with replace) (3, 4)**

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

Delete *object*

Object	Authorization required
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	Not applicable
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE
Communication information	MQZAO_DELETE

Inquire *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY
Communication information	MQZAO_DISPLAY

Inquire ***object*** **names**

Object	Authorization required
Queue	No check
Topic	No check
Process	No check
Queue manager	No check
Namelist	No check
Authentication information	No check
Channel	No check
Client connection channel	No check
Listener	No check
Service	No check
Communication information	No check

Start ***object***

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

Stop ***object***

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

Channel Commands

Command	Object	Authorization required
Ping Channel	Channel	MQZAO_CONTROL
Reset Channel	Channel	MQZAO_CONTROL_EXTENDED
Resolve Channel	Channel	MQZAO_CONTROL_EXTENDED

Subscription Commands

Command	Object	Authorization required
Change Subscription	Topic	MQZAO_CONTROL
Create Subscription	Topic	MQZAO_CONTROL
Delete Subscription	Topic	MQZAO_CONTROL
Inquire Subscription	Topic	MQZAO_DISPLAY

Security Commands

Command	Object	Authorization required
Set Authority Record	Queue manager	MQZAO_CHANGE
Delete Authority Record	Queue manager	MQZAO_CHANGE
Inquire Authority Records	Queue manager	MQZAO_DISPLAY
Inquire Authority Service	Queue manager	MQZAO_DISPLAY
Inquire Entity Authority	Queue manager	MQZAO_DISPLAY
Set Channel Authentication Record	Queue manager	MQZAO_CHANGE
Inquire Channel Authentication Records	Queue manager	MQZAO_DISPLAY
Refresh Security	Queue manager	MQZAO_CHANGE

Status Displays

Command	Object	Authorization required
Inquire Channel Status	Queue manager	MQZAO_DISPLAY Note that `+inq` authority (or equivalently MQZAO_INQUIRE) is required on the transmission queue if the channel type is CLUSSDR.
Inquire Channel Listener Status	Queue manager	MQZAO_DISPLAY
Inquire Pub/Sub Status	Queue manager	MQZAO_DISPLAY
Inquire Subscription Status	Queue manager	MQZAO_DISPLAY
Inquire Service Status	Queue manager	MQZAO_DISPLAY
Inquire Topic Status	Queue manager	MQZAO_DISPLAY

Cluster Commands

Command	Object	Authorization required
Inquire Cluster Queue Manager	Queue manager	MQZAO_DISPLAY
Refresh Cluster	'mqm' group membership required	'mqm' group membership required
Reset Cluster	'mqm' group membership required	'mqm' group membership required
Suspend Queue Manager Cluster	'mqm' group membership required	'mqm' group membership required
Resume Queue Manager Cluster	'mqm' group membership required	'mqm' group membership required

Other Administrative Commands

Command	Object	Authorization required
Ping Queue Manager	Queue manager	MQZAO_DISPLAY
Refresh Queue Manager	Queue manager	MQZAO_CHANGE
Reset Queue Manager	Queue manager	MQZAO_CHANGE
Reset Queue Statistics	Queue	MQZAO_DISPLAY and MQZAO_CHANGE
Inquire Connection	Queue manager	MQZAO_DISPLAY
Stop Connection	Queue manager	MQZAO_CHANGE

Note:
1. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the **setmqaut** command.
3. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
4. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

Special considerations for security on Windows

Some security functions behave differently on different versions of Windows.

IBM MQ security relies on calls to the operating system API for information about user authorizations and group memberships. Some functions do not behave identically on the Windows systems. This collection of topics includes descriptions of how those differences might affect IBM MQ security when you are running IBM MQ in a Windows environment.

The SSPI channel exit program

IBM MQ for Windows supplies a security exit program, which can be used on both message and MQI channels. The exit is supplied as source and object code, and provides one-way and two-way authentication.

The security exit uses the Security Support Provider Interface (SSPI), which provides the integrated security facilities of Windows platforms.

The security exit provides the following identification and authentication services:

One way authentication

This uses Windows NT LAN Manager (NTLM) authentication support. NTLM allows servers to authenticate their clients. It does not allow a client to authenticate a server, or one server to authenticate another. NTLM was designed for a network environment in which servers are assumed to be genuine. NTLM is supported on all Windows platforms that are supported by IBM WebSphere MQ Version 7.0.

This service is typically used on an MQI channel to enable a server queue manager to authenticate an IBM MQ MQI client application. A client application is identified by the user ID associated with the process that is running.

To perform the authentication, the security exit at the client end of a channel acquires an authentication token from NTLM and sends the token in a security message to its partner at the other end of the channel. The partner security exit passes the token to NTLM, which checks that the token is authentic. If the partner security exit is not satisfied with the authenticity of the token, it instructs the MCA to close the channel.

Two way, or mutual, authentication

This uses Kerberos authentication services. The Kerberos protocol does not assume that servers in a network environment are genuine. Servers can authenticate clients and other servers, and clients can authenticate servers. Kerberos is supported on all Windows platforms that are supported by IBM WebSphere MQ Version 7.0.

This service can be used on both message and MQI channels. On a message channel, it provides mutual authentication of the two queue managers. On an MQI channel, it enables the server queue manager and the IBM MQ MQI client application to authenticate each other. A queue manager is identified by its name prefixed by the string `ibmMQSeries/`. A client application is identified by the user ID associated with the process that is running.

To perform the mutual authentication, the initiating security exit acquires an authentication token from the Kerberos security server and sends the token in a security message to its partner. The partner security exit passes the token to the Kerberos server, which checks that it is authentic. The Kerberos security server generates a second token, which the partner sends in a security message to the initiating security exit. The initiating security exit then asks the Kerberos server to check that the second token is authentic. During this exchange, if either security exit is not satisfied with the authenticity of the token sent by the other, it instructs the MCA to close the channel.

The security exit is supplied in both source and object format. You can use the source code as a starting point for writing your own channel exit programs or you can use the object module as supplied. The object module has two entry points, one for one way authentication using NTLM authentication support and the other for two way authentication using Kerberos authentication services.

For more information about how the SSPI channel exit program works, and for instructions on how to implement it, see Using the SSPI security exit on Windows systems.

When you get a 'group not found' error on Windows

This problem can arise because IBM MQ loses access to the local mqm group when Windows servers are promoted to, or demoted from, domain controllers. To remedy this problem, re-create the local mqm group.

The symptom is an error indicating the lack of a local mqm group, for example:

```
>crtmqm qm0
AMQ8066:Local mqm group not found.
```

Altering the state of a machine between server and domain controller can affect the operation of IBM MQ, because IBM MQ uses a locally-defined mqm group. When a server is promoted to be a domain controller, the scope changes from local to domain local. When the machine is demoted to server, all domain local groups are removed. This means that changing a machine from server to domain controller and back to server loses access to a local mqm group.

To remedy this problem, re-create the local mqm group using the standard Windows management tools. Because all group membership information is lost, you must reinstate privileged IBM MQ users in the newly-created local mqm group. If the machine is a domain member, you must also add the domain mqm group to the local mqm group to grant privileged domain IBM MQ user IDs the required level of authority.

When you have problems with IBM MQ and domain controllers on Windows

Certain problems can arise with security settings when Windows servers are promoted to domain controllers.

While promoting Windows 2000, Windows 2003, or Windows Server 2008 servers to domain controllers, you are presented with the option of selecting a default or non-default security setting relating to user and group permissions. This option controls whether arbitrary users are able to retrieve group memberships from the active directory. Because IBM MQ relies on group membership information to implement its security policy, it is important that the user ID that is performing IBM MQ operations can determine the group memberships of other users.

On Windows 2000, when a domain is created using the default security option, the default user ID created by IBM MQ during the installation process can obtain group memberships for other users as required. The product then installs normally, creating default objects, and the queue manager can determine the access authority of local and domain users if required.

On Windows 2000, when a domain is created using the non-default security option, or on Windows 2003 and Windows Server 2008 when a domain is created using the default security option, the user ID created by IBM MQ during the installation cannot always determine the required group memberships. In this case, you need to know:

- How Windows 2000 with non-default, or Windows 2003 and Windows Server 2008 with default, security permissions behaves
- How to allow domain mqm group members to read group membership
- How to configure an IBM MQ Windows service to run under a domain user

Windows Server 2008 and Windows Server 2012 domain with default, security permissions:

Installation of IBM MQ behaves differently on these operating systems depending on whether a local user or domain user performs the installation.

If a **local** user installs IBM MQ, the Prepare IBM MQ Wizard detects that the local user created for the IBM MQ Windows service can retrieve the group membership information of the installing user. The Prepare IBM MQ Wizard asks the user questions about the network configuration to determine whether there are other user accounts defined on domain controllers running on Windows 2000 or later. If so, the IBM MQ Windows service needs to run under a domain user account with particular settings and authorities. The Prepare IBM MQ Wizard prompts the user for the account details of this user. Its online help provides details of the domain user account required that can be sent to the domain administrator.

If a **domain** user installs IBM MQ, the Prepare IBM MQ Wizard detects that the local user created for the IBM MQ Windows service cannot retrieve the group membership information of the installing user. In this case, the Prepare IBM MQ Wizard always prompts the user for the account details of the domain user account for the IBM MQ Windows service to use.

When the IBM MQ Windows service needs to use a domain user account, IBM MQ cannot operate correctly until this has been configured using the Prepare IBM MQ Wizard. The Prepare IBM MQ Wizard does not allow the user to continue with other tasks, until the Windows service has been configured with a suitable account.

See Creating and setting up domain accounts for IBM MQ for more information.

Configuring IBM MQ Services to run under a domain user on Windows:

Use the Prepare IBM MQ wizard to enter the account details of the domain user account. Alternatively, you can use the Computer Management panel to alter the **Log On** details for the installation specific IBM MQ Service.

For more information see Changing the password of the IBM MQ Windows service user account

Applying security template files to Windows

Applying a template might affect the security settings applied to IBM MQ files and directories. If you use the highly secure template, apply it before installing IBM MQ.

Windows supports text-based security template files that you can use to apply uniform security settings to one or more computers with the Security Configuration and Analysis MMC snap-in. In particular, Windows supplies several templates that include a range of security settings with the aim of providing specific levels of security. These templates include Compatible, Secure, and Highly Secure.

Applying one of these templates might affect the security settings applied to IBM MQ files and directories. If you want to use the Highly Secure template, configure your machine before you install IBM MQ.

If you apply the highly secure template to a machine on which IBM MQ is already installed, all the permissions you have set on the IBM MQ files and directories are removed. Because these permissions are removed, you lose *Administrator*, *mqm*, and, when applicable, *Everyone* group access from the error directories.

Nested groups

There are restrictions on the use of nested groups. These result partly from the domain functional level and partly from IBM MQ restrictions.

Active Directory can support different group types within a Domain context depending on the Domain functional level. By default, Windows 2003 domains are in the " Windows 2000 mixed" functional level. (Windows Server 2008 and Windows Server 2012 follow the Windows 2003 domain model.) The domain functional level determines the supported group types and level of nesting allowed when configuring user IDs in a domain environment. Refer to Active Directory documentation for details on the Group Scope and inclusion criteria.

In addition to Active Directory requirements, further restrictions are imposed on IDs used by IBM MQ. The network APIs used by IBM MQ do not support all the configurations that are supported by the domain functional level. As a result, IBM MQ is not able to query the group memberships of any Domain IDs present in a Domain Local group which is then nested in a local group. Furthermore, multiple nesting of global and universal groups is not supported. However, immediately nested global or universal groups are supported.

Configuring additional authority for Windows applications connecting to IBM MQ

The account under which IBM MQ processes run might need additional authorization before SYNCHRONIZE access to application processes can be granted.

You might experience problems if you have Windows applications, for example ASP pages, connecting to IBM MQ that are configured to run at a security level higher than usual.

IBM MQ requires SYNCHRONIZE access to application processes in order to coordinate certain actions. APAR IC35116 changed IBM MQ so that the appropriate privileges are specified. However, the account under which IBM MQ processes run might need additional authorization before the requested access can be granted.

When a server application first attempts to connect to a queue manager IBM MQ will modify the process to grant SYNCHRONIZE authority for IBM MQ administrators. To configure additional authority to the user ID under which IBM MQ processes are running, complete the following steps:

1. Start the Local Security Policy tool, click Security Settings ->Local Policies->User Right Assignments, click "Debug Programs".
2. Double click "Debug Programs", then add your IBM MQ user ID to the list

If the system is in a Windows domain and the effective policy setting is still not set, even though the local policy setting is set, the user ID must be authorized in the same way at domain level, using the Domain Security Policy tool.

Setting up security on HP Integrity NonStop Server

Security considerations specific to HP Integrity NonStop Server systems.

The IBM MQ client for HP Integrity NonStop Server supports both the Transport Layer Security (TLS) and the Secure Sockets Layer (SSL) protocols to provide link level security when you are connecting to a queue manager. These protocols are supported by using an implementation of OpenSSL. OpenSSL requires a source of random data for providing strong cryptographic operations.

OpenSSL

OpenSSL security overview for IBM MQ client for HP Integrity NonStop Server.

The OpenSSL toolkit is an open source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols for secure communications over a network.

The toolkit is developed by the OpenSSL Project. For more information about the OpenSSL Project, see http://www.openssl.org. IBM MQ client for HP Integrity NonStop Server contains modified versions of the OpenSSL libraries and the **openssl** command. The libraries and **openssl** command are ported from the OpenSSL toolkit 1.0.1c, and are supplied as object code only. No source code is provided.

The OpenSSL libraries are loaded by IBM MQ client application programs dynamically as required. Only the OpenSSL libraries that are provided by IBM MQ are supported for use with IBM MQ client applications.

The **openssl** command, which can be used for certificate management purposes, is installed in the OSS directory `opt_installation_path/opt/mqm/bin`.

Using the **openssl** command, you can create and manage keys and digital certificates with various common data formats, and carry out simple certificate authority (CA) tasks.

The default format for key and certificate data that is processed by OpenSSL is the Privacy Enhanced Mail (PEM) format. Data in PEM format is base64 encoded ASCII data. The data can therefore be transferred by using text-based systems such as email, and can be cut and pasted by using text editors and web browsers. PEM is an Internet standard for text-based cryptographic exchanges and is specified in Internet RFCs 1421, 1422, 1423, and 1424. IBM MQ assumes that a file with extension `.pem` contains data in PEM format. A file in PEM format can contain multiple certificates and other encoded objects, and can include comments.

The IBM MQ SSL support on other operating systems might require key and certificate data in files to be encoded by using Distinguished Encoding Rules (DER). DER is a set of encoding rules for using the ASN.1 notation in secure communications. Data that is encoded by using DER is binary data, and the format of key and certificate data that is encoded by using DER is also known as PKCS#12 or PFX. A file that contains this data commonly has an extension of `.p12` or `.pfx`. The **openssl** command can convert between PEM and PKCS#12 format.

Entropy Daemon

OpenSSL requires a source of random data for providing strong cryptographic operations. Random number generation is a capability that is usually provided by the operating system or by a system-wide daemon process. The HP Integrity NonStop Server operating system does not provide this capability within the operating system.

When you are using the SSL and TLS support supplied with IBM MQ client for HP Integrity NonStop Server, a process that is called an entropy daemon is needed to provide the source of random data. When you start a client channel that requires SSL or TLS, OpenSSL expects an entropy daemon to be running and providing its services on a socket in the OSS file system at `/etc/egd-pool`.

An entropy daemon is not provided by IBM MQ client for HP Integrity NonStop Server. The IBM MQ client for HP Integrity NonStop Server is tested with the following entropy daemons:

- amqjkdm0 (as provided by the IBM MQ 5.3 server)
- `/usr/local/bin/prngd` (Version 0.9.27, as provided by HP Integrity NonStop Server Open Source Technical Library)

Setting up security on IBM i

Security for IBM MQ for IBM i is implemented using the IBM MQ Object Authority Manager (OAM) and IBM i object level security.

Security considerations that must be made when determining access authority to IBM MQ objects.

You need to consider the following points when setting up authorities to the users in your enterprise:

1. Grant and revoke authorities to the IBM MQ for IBM i commands using the IBM i **GRTOBJAUT** and **RVKOBJAUT** commands.

   In the `QMQM` library, certain noncommand (*cmd) objects are set to have **`*PUBLIC`** authority to **`*USE`**. Do not change the authorities of these objects or use an authorization list to provide authority. Any incorrect authority might compromise IBM MQ functionality.
2. During installation of IBM MQ for IBM i, the following special user profiles are created:

   QMQM
   : Is used primarily for internal product-only functions. However, it can be used to run trusted applications using MQCNO_FASTPATH_BINDINGS. See Connecting to a queue manager using the MQCONNX call.

   QMQMADM
   : Is used as a group profile for administrators of IBM MQ. The group profile gives access to CL commands and IBM MQ resources.

   When using SBMJOB to submit programs that call IBM MQ commands, USER must not be set explicitly to QMQMADM. Instead, set USER to QMQM or another user profile that has QMQMADM specified as a group.
3. If you are sending channel commands to remote queue managers, ensure that your user profile is a member of the group QMQMADM on the target system. For a list of PCF and MQSC channel commands, see IBM MQ for IBM i CL commands.
4. The group set associated with a user is cached when the group authorizations are computed by the OAM.

   Any changes made to a user's group memberships after the group set has been cached are not recognized until you restart the queue manager or execute RFRMQMAUT to refresh security.
5. Limit the number of users who have authority to work with commands that are particularly sensitive. These commands include:
   - Create Message Queue Manager (**CRTMQM**)

- Delete Message Queue Manager (**DLTMQM**)
- Start Message Queue Manager (**STRMQM**)
- End Message Queue Manager (**ENDMQM**)
- Start Command Server (**STRMQMCSVR**)
- End Command Server (**ENDMQMCSVR**)

6. Channel definitions contain a security exit program specification. Channel creation and modification requires special considerations. Details of security exits are given in “Security exit overview” on page 477.
7. The channel exit and trigger monitor programs can be substituted. The security of such replacements is the responsibility of the programmer.

The object authority manager on IBM i

The object authority manager (OAM) manages users' authorizations to manipulate IBM MQ objects, including queues and process definitions. It also provides a command interface through which you can grant or revoke access authority to an object for a specific group of users. The decision to allow access to a resource is made by the OAM, and the queue manager follows that decision. If the OAM cannot make a decision, the queue manager prevents access to that resource.

Through the OAM you can control:

- Access to IBM MQ objects through the MQI. When an application program attempts to access an object, the OAM checks that the user profile making the request has the authorization for the operation requested.

  In particular, this means that queues, and the messages on queues, can be protected from unauthorized access.
- Permission to use PCF and MQSC commands.

Different groups of users can have different access authority to the same object. For example, for a specific queue, one group could perform both put and get operations; another group might be allowed only to browse the queue (MQGET with browse option). Similarly, some groups might have get and put authority to a queue, but not be allowed to alter or delete the queue.

IBM MQ for IBM i commands and perform operations on IBM MQ for IBM i objects

IBM MQ authorities on IBM i

To access IBM MQ objects, you need authority to issue the command and to access the object referenced. Administrators have access to all IBM MQ resources.

Access to IBM MQ objects is controlled by authorities to:

1. Issue the IBM MQ command
2. Access the IBM MQ objects referenced by the command

All IBM MQ for IBM i CL commands are shipped with an owner of QMQM, and the administration profile (QMQMADM) has *USE rights with the *PUBLIC access set to *EXCLUDE.

Note: The QSRDUPER program is used by the IBM MQ for IBM i licensed program install to duplicate Command (*CMD) objects in QSYS. In IBM i V5R4 and later, the QSRDUPER program was changed so that the default behavior is to create a proxy command rather than a duplicate of the original command. A proxy command redirects command execution to another command and has an attribute of PRX. If a proxy command by the same name as the command being copied exists in library QSYS, private authorities to the proxy command are not granted to the command in the product library. Attempts to prompt or run the proxy command in QSYS check the authority of the target command in the product library. Any changes in authority to *CMD objects therefore need to be done in the product library (QMQM) and those in QSYS do not need to be modified. For example:

```
GRTOBJAUT OBJ(QMQM/DSPMQMQ) OBJTYPE(*CMD) USER(MQUSER) AUT(*USE)
```

Changes to the authority structure of some of the product's CL commands allows public use of these commands, if you have the required OAM authority to the IBM MQ objects to make these changes.

To be an IBM MQ administrator on IBM i, you must be a member of the *QMQMADM group*. This group has properties like the properties of the mqm group on UNIX, Linux and Windows systems. In particular, the QMQMADM group is created when you install IBM MQ for IBM i, and members of the QMQMADM group have access to all IBM MQ resources on the system. You also have access to all IBM MQ resources if you have *ALLOBJ authority.

Administrators can use CL commands to administer IBM MQ. One of these commands is GRTMQMAUT, which is used to grant authorities to other users. Another command, STRMQMMQSC, enables an administrator to issue MQSC commands to a local queue manager.

Related concepts:

"Authority to administer IBM MQ on IBM i" on page 449

Access authorities for IBM MQ objects on IBM i

Access authorities required for running IBM MQ CL commands.

IBM MQ for IBM i categorizes the product's CL commands into two groups:

Group 1

Users must be in the QMQMADM user group, or have *ALLOBJ authority, to process these commands. Users having either of these authorities can process all commands in all categories without requiring any extra authority.

Note: These authorities override any OAM authority.

These commands can be grouped as follows:

- Command Server Commands
  - ENDMQMCSVR, End IBM MQ Command Server
  - STRMQMCSVR, Start IBM MQ Command Server
- Dead-Letter Queue Handler Command
  - STRMQMDLQ, Start IBM MQ Dead-Letter Queue Handler
- Listener Command
  - ENDMQMLSR, End IBM MQ listener
  - STRMQMLSR, Start non-object listener
- Media Recovery Commands
  - RCDMQMIMG, Record IBM MQ Object Image
  - RCRMQMOBJ, Re-create IBM MQ Object
  - WRKMQMTRN, Work with IBM MQ Q Transactions
- Queue Manager Commands
  - CRTMQM, Create Message Queue Manager
  - DLTMQM, Delete Message Queue Manager
  - ENDMQM, End Message Queue Manager
  - STRMQM, Start Message Queue Manager
- Security Commands
  - GRTMQMAUT, Grant IBM MQ Object Authority

  - RVKMQMAUT, Revoke IBM MQ Object Authority
- Trace Command
  - TRCMQM, Trace IBM MQ Job
- Transaction Commands
  - RSVMQMTRN, Resolve IBM MQ Transaction
- Trigger Monitor Commands
  - STRMQMTRM, Start Trigger Monitor
- IBM MQSC Commands
  - RUNMQSC, Run IBM MQSC Commands
  - STRMQMMQSC, Start IBM MQSC Commands

Group 2

The rest of the commands, for which two levels of authority are required:

1. IBM i authority to run the command. An IBM MQ administrator sets this using the **GRTOBJAUT** command to override the *PUBLIC(*EXCLUDE) restriction for a user or group of users.

   For example:

   ```
   GRTOBJAUT OBJ(QMQM/DSPMQMQ) OBJTYPE(*CMD) USER(MQUSER) AUT(*USE)
   ```
2. IBM MQ authority to manipulate the IBM MQ objects associated with the command, or commands, given the correct IBM i authority in Step 1.

   This authority is controlled by the user having the appropriate OAM authority for the required action, set by an IBM MQ administrator using the **GRTMQMAUT** command

   For example:

   ```
   GRTMQMAUT *connect authority to the queue manager + *admchg authority to
     the queue
   ```

The commands can be grouped as follows:

- Channel Commands
  - CHGMQMCHL, Change IBM MQ Channel

    This requires *connect authority to the queue manager and *admchg authority to the channel.
  - CPYMQMCHL, Copy IBM MQ Channel

    This requires *connect and *admcrt authority to the queue manager, *admdsp authority to the default channel type to be copied, and *admcrt authority to the channel object class.

    For example, copying a Sender channel, needs *admdsp authority to SYSTEM.DEF.SENDER channel
  - CRTMQMCHL, Create IBM MQ Channel

    This requires *connect and *admcrt authority to the queue manager, *admdsp authority to the default channel type to be created and *admcrt authority to the channel object class.

    For example, creating a Sender channel, needs *admdsp authority to SYSTEM.DEF.SENDER channel
  - DLTMQMCHL, Delete IBM MQ Channel

    This requires *connect authority to the queue manager and *admdlt authority to the channel.
  - RSVMQMCHL, Resolve IBM MQ Channel

    This requires *connect authority to the queue manager and *ctrlx authority to the channel.
- Display commands

  To process the DSP commands you must grant the user `*connect` and `*admdsp` authority to the queue manager, together with any specific option listed:
  - DSPMQM, Display Message Queue Manager

- DSPMQMAUT, Display IBM MQ Object Authority
- DSPMQMAUTI, Display IBM MQ Authentication Information - *admdsp to the authentication information object
- DSPMQMCHL, Display IBM MQ Channel - *admdsp to the channel
- DSPMQMCSVR, Display IBM MQ Command Server
- DSPMQMNL, Display IBM MQ Namelist - *admdsp to the namelist
- DSPMQMOBJN, Display IBM MQ Object Names
- DSPMQMPRC, Display IBM MQ Process - *admdsp to the process
- DSPMQMQ, Display IBM MQ Queue - *admdsp to the queue
- DSPMQMTOP, Display IBM MQ Topic - *admdsp to the topic

• Work with commands

  To process the WRK commands and display the options panel you must grant the user *connect and *admdsp authority to the queue manager, together with any specific option listed:
  - WRKMQM, Work with Message Queue Managers
  - WRKMQMAUT, Work with IBM MQ Object Authority
  - WRKMQMAUTD, Work with IBM MQ Object Authority Data
  - WRKMQMAUTI, Work with IBM MQ Authentication Information
    - *admchg for the Change IBM MQ Authentication Information Object command.
    - *admcrt for the Create and Copy IBM MQ Authentication Information Object command.
    - *admdlt for the Delete IBM MQ Authentication Information Object command.
    - *admdsp for the Display IBM MQ Authentication Information Object command.
  - WRKMQMCHL, Work with IBM MQ Channel

    This requires the following authorities:
    - *admchg for the Change IBM MQ Channel command.
    - *admclr for the Clear IBM MQ Channel command.
    - *admcrt for the Create and Copy IBM MQ Channel command.
    - *admdlt for the Delete IBM MQ Channel command.
    - *admdsp for the Display IBM MQ Channel command.
    - *ctrl for the Start IBM MQ Channel command.
    - *ctrl for the End IBM MQ Channel command.
    - *ctrl for the Ping IBM MQ Channel command.
    - *ctrlx for the Reset IBM MQ Channel command.
    - *ctrlx for the Resolve IBM MQ Channel command.
  - WRKMQMCHST, Work with IBM MQ Channel Status

    This requires *admdsp authority to the channel.
  - WRKMQMCL, Work with IBM MQ Clusters
  - WRKMQMCLQ, Work with IBM MQ Cluster Queues
  - WRKMQMCLQM, Work with IBM MQ Cluster Queue Manager
  - WRKMQMLSR, Work with IBM MQ Listener
  - WRKMQMMSG, Work with IBM MQ Messages

    This requires *browse authority to the queue
  - WRKMQMNL, Work with IBM MQ Namelists

    This requires the following authorities:
    - *admchg for the Change IBM MQ Namelist command.
    - *admcrt for the Create and Copy IBM MQ Namelist command.

    - `*admdlt` for the Delete IBM MQ Namelist command.
    - `*admdsp` for the Display IBM MQ Namelist command.
  - WRKMQMPRC, Work with IBM MQ Processes

    This requires the following authorities:
    - `*admchg` for the Change IBM MQ Process command.
    - `*admcrt` for the Create and Copy IBM MQ Process command.
    - `*admdlt` for the Delete IBM MQ Process command.
    - `*admdsp` for the Display IBM MQ Process command.
  - WRKMQMQ, Work with IBM MQ queues

    This requires the following authorities:
    - `*admchg` for the Change IBM MQ Queue command.
    - `*admclr` for the Clear IBM MQ Queue command.
    - `*admcrt` for the Create and Copy IBM MQ Queue command.
    - `*admdlt` for the Delete IBM MQ Queue command.
    - `*admdsp` for the Display IBM MQ Queue command.
  - WRKMQMQSTS, Work with IBM MQ Queue Status
  - WRKMQMTOP, Work with IBM MQ Topics

    This requires the following authorities
    - `*admchg` for the Change IBM MQ Topic command.
    - `*admcrt` for the Create and Copy IBM MQ Topic command.
    - `*admdlt` for the Delete IBM MQ Topic command.
    - `*admdsp` for the Display IBM MQ Topic command.
  - WRKMQMSUB, Work with IBM MQ Subscriptions
- Other Channel commands

  To process the channel commands you must grant the user the specific authorities listed:
  - ENDMQMCHL, End IBM MQ Channel

    This requires `*connect` authority to the queue manager and `*allmqi` authority to the transmission queue associated with the channel.
  - ENDMQMLSR, End IBM MQ Listener

    This requires `*connect` authority to the queue manager and `*ctrl` authority to the named listener object.
  - PNGMQMCHL, Ping IBM MQ Channel

    This requires `*connect` and `*inq` authority to the queue manager and `*ctrl` authority to the channel object.
  - RSTMQMCHL, Reset IBM MQ Channel

    This requires `*connect` authority to the queue manager.
  - STRMQMCHL, Start IBM MQ Channel

    This requires `*connect` authority to the queue manager and `*ctrl` authority to the channel object.
  - STRMQMCHLI, Start IBM MQ Channel Initiator

    This requires `*connect` and `*inq` authority to the queue manager, and `*allmqi` authority to the initiation queue associated with the transmission queue of the channel.
  - STRMQMLSR, Start IBM MQ Listener

    This requires *connect authority to the queue manager and *ctrl authority to the named listener object.
- Other commands:

To process the following commands you must grant the user the specific authorities listed:

- CCTMQM, Connect to Message Queue Manager

  This requires no IBM MQ object authority.
- CHGMQM, Change Message Queue Manager

  This requires `*connect` and `*admchg` authority to the queue manager.
- CHGMQMAUTI, Change IBM MQ Authentication Information

  This requires `*connect` authority to the queue manager and `*admchg` and `*admdsp` authority to the authentication information object.
- CHGMQMNL, Change IBM MQ Namelist

  This requires `*connect` authority to the queue manager and `*admchg` authority to the namelist.
- CHGMQMPRC, Change IBM MQ Process

  This requires `*connect` authority to the queue manager and `*admchg` authority to the process.
- CHGMQMQ, Change IBM MQ Queue

  This requires `*connect` authority to the queue manager and `*admchg` authority to the queue.
- CLRMQMQ, Clear IBM MQ Queue

  This requires `*connect` authority to the queue manager and `*admclr` authority to the queue.
- CPYMQMAUTI, Copy IBM MQ Authentication Information

  This requires `*connect` authority to the queue manager and `*admdsp` authority to the authentication information object and `*admcrt` authority to the authentication information object class.
- CPYMQMNL, Copy IBM MQ Namelist

  This requires `*connect` and `*admcrt` authority to the queue manager.
- CPYMQMPRC, Copy IBM MQ Process

  This requires `*connect` and `*admcrt` authority to the queue manager.
- CPYMQMQ, Copy IBM MQ Queue

  This requires `*connect` and `*admcrt` authority to the queue manager.
- CRTMQMAUTI, Create IBM MQ Authentication Information

  This requires `*connect` authority to the queue manager and `*admdsp` authority to the authentication information object and `*admcrt` authority to the authentication information object class.
- CRTMQMNL, Create IBM MQ Namelist

  This requires `*connect` and `*admcrt` authority to the queue manager and `*admdsp` authority to the default namelist.
- CRTMQMPRC, Create IBM MQ Process

  This requires `*connect` and `*admcrt` authority to the queue manager and `*admdsp` authority to the default process.
- CRTMQMQ, Create IBM MQ Queue

  This requires `*connect` and `*admcrt` authority to the queue manager and `*admdsp` authority to the default queue.
- CVTMQMDTA, Convert IBM MQ Data Type Command

  This requires no IBM MQ object authority.
- DLTMQMAUTI, Delete IBM MQ Authentication Information

  This requires `*connect` authority to the queue manager and `*ctrlx` authority to the authentication information object.
- DLTMQMNL, Delete IBM MQ Namelist

This requires `*connect` authority to the queue manager and `*admdlt` authority to the namelist.
- DLTMQMPRC, Delete IBM MQ Process

  This requires `*connect` authority to the queue manager and `*admdlt` authority to the process.
- DLTMQMQ, Delete IBM MQ Queue

  This requires `*connect` authority to the queue manager and `*admdlt` authority to the queue.
- DSCMQM, Disconnect from Message Queue Manager

  This requires no IBM MQ object authority.
- RFRMQMAUT, Refresh Security

  This requires `*connect` authority to the queue manager.
- RFRMQMCL, Refresh Cluster

  This requires `*connect` authority to the queue manager.
- RSMMQMCLQM, Resume Cluster Queue Manager

  This requires `*connect` authority to the queue manager.
- RSTMQMCL, Reset Cluster

  This requires `*connect` authority to the queue manager.
- SPDMQMCLQM, Suspend Cluster Queue Manager

  This requires `*connect` authority to the queue manager.

Access authorizations on IBM i

Use this information to understand the access authorization commands.

Authorizations defined by the AUT keyword on the **GRTMQMAUT** and **RVKMQMAUT** commands can be categorized as follows:
- Authorizations related to MQI calls
- Authorization-related administration commands
- Context authorizations
- General authorizations, that is, for MQI calls, for commands, or both

The following tables list the different authorities, using the AUT parameter for MQI calls, Context calls, MQSC and PCF commands, and generic operations.

Table 27. Authorizations for MQI calls

AUT	Description
***ALTUSR**	Allow another user's authority to be used for MQOPEN and MQPUT1 calls.
***BROWSE**	Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.
***CONNECT**	Connect the application to the specified queue manager by issuing an MQCONN call.
***GET**	Retrieve a message from a queue by issuing an MQGET call.
***INQ**	Make an inquiry on a specific queue by issuing an MQINQ call.
***PUB**	Open a topic to publish a message using an MQPUT call.
***PUT**	Put a message on a specific queue by issuing an MQPUT call.
***RESUME**	Resume a subscription using an MQSUB call.
***SET**	Set attributes on a queue from the MQI by issuing an MQSET call. If you open a queue for multiple options, you must be authorized for each of them.
***SUB**	Create, Alter or Resume a subscription to a topic using an MQSUB call.

Table 28. Authorizations for context calls

AUT	Description
***PASSALL**	Pass all context on the specified queue. All the context fields are copied from the original request.
***PASSID**	Pass identity context on the specified queue. The identity context is the same as that of the request.
***SETALL**	Set all context on the specified queue. This is used by special system utilities.
***SETID**	Set identity context on the specified queue. This is used by special system utilities.

Table 29. Authorizations for MQSC and PCF calls

AUT	Description
***ADMCHG**	Change the attributes of the specified object.
***ADMCLR**	Clear the specified object (PCF Clear object command only).
***ADMCRT**	Create objects of the specified type.
***ADMDLT**	Delete the specified object.
***ADMDSP**	Display the attributes of the specified object.

Table 30. Authorizations for generic operations

AUT	Description
***ALL**	Use all operations applicable to the object. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.
***ALLADM**	Perform all administration operations applicable to the object.
***ALLMQI**	Use all MQI calls applicable to the object.
***CTRL**	Control startup and shutdown of channels, listeners, and services.
***CTRLX**	Reset sequence number and resolve indoubt channels.

Using the access authorization commands on IBM i

Use this information to learn about the access authorization commands, and use the command examples.

Using the GRTMQMAUT command

If you have the required authorization, you can use the **GRTMQMAUT** command to grant authorization of a user profile or user group to access a particular object. The following examples illustrate how the **GRTMQMAUT** command is used:

1.
   ```
   GRTMQMAUT OBJ(RED.LOCAL.QUEUE) OBJTYPE(*LCLQ) USER(GROUPA) +
             AUT(*BROWSE *PUT) MQMNAME('saturn.queue.manager')
   ```

   In this example:

   - `RED.LOCAL.QUEUE` is the object name.
   - `*LCLQ` (local queue) is the object type.
   - `GROUPA` is the name of a user profile on the system for which authorizations are to change. This profile can be used as a group profile for other users.
   - `*BROWSE` and `*PUT` are the authorizations being granted to the specified queue.

     `*BROWSE` adds authorization to browse messages on the queue (to issue MQGET with the browse option).

     `*PUT` adds authorization to put (MQPUT) messages on the queue.
   - `saturn.queue.manager` is the queue manager name.

2. The following command grants to users JACK and JILL all applicable authorizations, to all process definitions, for the default queue manager.

   ```
   GRTMQMAUT OBJ(*ALL) OBJTYPE(*PRC) USER(JACK JILL) AUT(*ALL)
   ```
3. The following command grants user GEORGE authority to put a message on the queue ORDERS, on the queue manager TRENT.

   ```
   GRTMQMAUT OBJ(TRENT) OBJTYPE(*MQM) USER(GEORGE) AUT(*CONNECT) MQMNAME (TRENT)
   GRTMQMAUT OBJ(ORDERS) OBJTYPE(*Q) USER(GEORGE) AUT(*PUT) MQMNAME (TRENT)
   ```

Using the RVKMQMAUT command

If you have the required authorization, you can use the **RVKMQMAUT** command to remove previously granted authorization of a user profile or user group to access a particular object. The following examples illustrate how the **RVKMQMAUT** command is used:

1.
   ```
   RVKMQMAUT OBJ(RED.LOCAL.QUEUE) OBJTYPE(*LCLQ) USER(GROUPA) +
   AUT(*PUT) MQMNAME('saturn.queue.manager')
   ```

   The authority to put messages to the specified queue, that was granted in the previous example, is removed for GROUPA.

2.
   ```
   RVKMQMAUT OBJ(PAY*) OBJTYPE(*Q) USER(*PUBLIC) AUT(*GET) +
   MQMNAME(PAYROLLQM)
   ```

   Authority to get messages from any queue with a name starting with the characters PAY, owned by queue manager PAYROLLQM, is removed from all users of the system unless they, or a group to which they belong, have been separately authorized.

Using the DSPMQMAUT command

The display MQM authority (**DSPMQMAUT**) command shows, for the specified object and user, the list of authorizations that the user has for the object. The following example illustrates how the command is used:

```
DSPMQMAUT OBJ(ADMINNL) OBJTYPE(*NMLIST) USER(JOE) OUTPUT(*PRINT) +
MQMNAME(ADMINQM)
```

Using the RFRMQMAUT command

The refresh MQM security (**RFRMQMAUT**) command enables you to update the OAM's authorization group information immediately, reflecting changes made at the operating system level, without needing to stop and restart the queue manager. The following example illustrates how the command is used:

```
RFRMQMAUT MQMNAME(ADMINQM)
```

Authorization specification tables for IBM i

Use this information to determine what authorization is required to use particular API calls, and particular options of those calls, on queue objects, process objects, and queue manager objects.

The authorization specification tables starting in Table 31 on page 549 define precisely how the authorizations work and the restrictions that apply. The tables apply to these situations:
- Applications that issue MQI calls
- Administration programs that issue MQSC commands as escape PCFs
- Administration programs that issue PCF commands

In this section, the information is presented as a set of tables that specify the following data:

Action to be performed
MQI option, MQSC command, or PCF command.

Access control object
Queue, process definition, queue manager, namelist, channel, client connection channel, listener, service, or authentication information object.

Authorization required
Expressed as an MQZAO_ constant.

In the tables, the constants prefixed by MQZAO_ correspond to the keywords in the authorization list for the **GRTMQMAUT** and **RVKMQMAUT** commands for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword *BROWSE ; similarly, the keyword MQZAO_SET_ALL_CONTEXT corresponds to the keyword *SETALL, and so on. These constants are defined in the header file cmqzc.h, which is supplied with the product.

MQI authorizations

An application is allowed to issue specific MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls require authorization checks: MQCONN, MQOPEN, MQPUT1, and MQCLOSE.

For MQOPEN and MQPUT1, the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application can be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of name resolution that is not a queue-manager alias, unless the queue-manager alias definition is opened directly; that is, its name appears in the *ObjectName* field of the object descriptor. Authority is always needed for the particular object being opened; in some cases additional queue-independent authority, obtained through an authorization for the queue-manager object, is required.

Table 31 on page 549, Table 32 on page 549, Table 33 on page 549, and Table 34 on page 550 summarize the authorizations needed for each call.

Note: These tables do not mention namelists, channels, client connection channels, listeners, services, or authentication information objects. This is because none of the authorizations apply to these objects, except for MQOO_INQUIRE, for which the same authorizations apply as for the other objects.

Table 31. Security authorization needed for MQCONN calls

Authorization required for:	Queue object (**1 on page 550**)	Process object	Queue manager object
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT

Table 32. Security authorization needed for MQOPEN calls

Authorization required for:	Queue object (**1 on page 550**)	Process object	Queue manager object
MQOO_INQUIRE	MQZAO_INQUIRE **(2 on page 550)**	MQZAO_INQUIRE **(2 on page 550)**	MQZAO_INQUIRE **(2 on page 550)**
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT **(3 on page 550)**	MQZAO_INPUT	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) **(4 on page 550)**	MQZAO_OUTPUT	Not applicable	Not applicable
MQOO_PASS_ IDENTITY_CONTEXT **(5 on page 550)**	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ALL_ CONTEXT **(5 on page 550, 6 on page 550)**	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT **(5 on page 550, 6 on page 550)**	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(7 on page 550)**
MQOO_SET_ ALL_CONTEXT **(5 on page 550, 8 on page 550)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7 on page 550)**
MQOO_OUTPUT (Transmission queue) **(9 on page 550)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7 on page 550)**
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	**(10 on page 550)**	**(10 on page 550)**	MQZAO_ALTERNATE_ USER_AUTHORITY **(10 on page 550, 11 on page 550)**

Table 33. Security authorization needed for MQPUT1 calls

Authorization required for:	Queue object (**1 on page 550**)	Process object	Queue manager object
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_CONTEXT **(12 on page 550)**	Not applicable	No check
MQPMO_PASS_ALL _CONTEXT	MQZAO_PASS_ ALL_CONTEXT **(12 on page 550)**	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_CONTEXT **(12 on page 550)**	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(7 on page 550)**

Table 33. Security authorization needed for MQPUT1 calls (continued)

Authorization required for:	**Queue object (1)**	**Process object**	**Queue manager object**
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT **(12)**	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**
(Transmission queue) **(9)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(7)**
MQPMO_ALTERNATE_ USER_AUTHORITY	**(13)**	Not applicable	MQZAO_ALTERNATE_ USER_AUTHORITY **(11)**

Table 34. Security authorization needed for MQCLOSE calls

Authorization required for:	**Queue object (1)**	**Process object**	**Queue manager object**
MQCO_DELETE	MQZAO_DELETE **(14)**	Not applicable	Not applicable
MQCO_DELETE _PURGE	MQZAO_DELETE **(14)**	Not applicable	Not applicable

Notes for the tables:

1. If a model queue is being opened:
   - MQZAO_DISPLAY authority is needed for the model queue, in addition to the authority to open the model queue for the type of access for which you are opening.
   - MQZAO_CREATE authority is not needed to create the dynamic queue.
   - The user identifier used to open the model queue is automatically granted all the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. Either the queue, process, namelist, or queue manager object is checked, depending on the type of object being opened.
3. MQOO_INPUT_* must also be specified. This option is valid for a local, model, or alias queue.
4. This check is performed for all output cases, except the case specified in note 9.
5. MQOO_OUTPUT must also be specified.
6. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
7. This authority is required for both the queue manager object and the particular queue.
8. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
9. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
10. At least one of MQOO_INQUIRE (for any object type), or (for queues) MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET must also be specified. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
11. This authorization allows any *AlternateUserId* to be specified.
12. An MQZAO_OUTPUT check is also carried out if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
13. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
14. The check is carried out only if both of the following are true:

- A permanent dynamic queue is being closed and deleted.
- The queue was not created by the MQOPEN that returned the object handle being used.

Otherwise, there is no check.

General notes:

1. The special authorization MQZAO_ALL_MQI includes all the following authorizations that are relevant to the object type:
   - MQZAO_CONNECT
   - MQZAO_INQUIRE
   - MQZAO_SET
   - MQZAO_BROWSE
   - MQZAO_INPUT
   - MQZAO_OUTPUT
   - MQZAO_PASS_IDENTITY_CONTEXT
   - MQZAO_PASS_ALL_CONTEXT
   - MQZAO_SET_IDENTITY_CONTEXT
   - MQZAO_SET_ALL_CONTEXT
   - MQZAO_ALTERNATE_USER_AUTHORITY
2. MQZAO_DELETE (see note 14 on page 550) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
3. *No check* means that no authorization checking is carried out.
4. *Not applicable* means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

Authorizations for MQSC commands in escape PCFs on IBM i

These authorizations allow a user to issue administration commands as an escape PCF message. These methods allow a program to send an administration command as a message to a queue manager, for execution on behalf of that user.

This section summarizes the authorizations needed for each MQSC command contained in Escape PCF.

Not applicable means that authorization checking is not relevant to this operation.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands
- Authority to issue the MQSC commands within the text of the Escape PCF command

ALTER *object*

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

CLEAR *object*

Object	Authorization required
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

DEFINE *object* **NOREPLACE (1 on page 555)**

Object	Authorization required
Queue	MQZAO_CREATE **(2 on page 555)**
Topic	MQZAO_CREATE **(2 on page 555)**
Process	MQZAO_CREATE **(2 on page 555)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(2 on page 555)**
Authentication information	MQZAO_CREATE **(2 on page 555)**
Channel	MQZAO_CREATE **(2 on page 555)**
Client connection channel	MQZAO_CREATE **(2 on page 555)**
Listener	MQZAO_CREATE **(2 on page 555)**
Service	MQZAO_CREATE **(2 on page 555)**

DEFINE *object* **REPLACE (1 on page 555, 3 on page 555)**

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

DELETE *object*

Object	Authorization required
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	Not applicable
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE

DISPLAY *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	
Service	

PING CHANNEL

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

RESET CHANNEL

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

RESOLVE CHANNEL

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

START *object*

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

STOP *object*

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

Note:
1. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the **GRTMQMAUT** command.
3. This option applies if the object to be replaced already exists. If it does not, the check is as for DEFINE *object* NOREPLACE.

Authorizations for PCF commands on IBM i

These authorizations allow a user to issue administration commands as PCF commands. These methods allow a program to send an administration command as a message to a queue manager, for execution on behalf of that user.

This section summarizes the authorizations needed for each PCF command.

No check means that no authorization checking is carried out; *Not applicable* means that authorization checking is not relevant to this operation.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands

The special authorization MQZAO_ALL_ADMIN includes the following authorizations:

- MQZAO_CHANGE
- MQZAO_CLEAR
- MQZAO_DELETE
- MQZAO_DISPLAY
- MQZAO_CONTROL
- MQZAO_CONTROL_EXTENDED

MQZAO_CREATE is not included as it is not specific to a particular object or object type

`Change` *`object`*

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

`Clear` *`object`*

Object	Authorization required
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Copy *object* (without replace) (1 on page 561)

Object	Authorization required
Queue	MQZAO_CREATE **(2 on page 561)**
Topic	MQZAO_CREATE **(2 on page 561)**
Process	MQZAO_CREATE **(2 on page 561)**
Queue manager	Not applicable
NamelistMQZAO_CREATE	MQZAO_CREATE **(2 on page 561)**
Authentication information	MQZAO_CREATE **(2 on page 561)**
Channel	MQZAO_CREATE **(2 on page 561)**
Client connection channel	MQZAO_CREATE **(2 on page 561)**
Listener	MQZAO_CREATE **(2 on page 561)**
Service	MQZAO_CREATE **(2 on page 561)**

Copy *object* (with replace) (1 on page 561, 4 on page 561)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Create *object* (without replace) (3 on page 561)

Object	Authorization required
Queue	MQZAO_CREATE **(2 on page 561)**
Topic	MQZAO_CREATE **(2 on page 561)**
Process	MQZAO_CREATE **(2 on page 561)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(2 on page 561)**
Authentication information	MQZAO_CREATE **(2 on page 561)**
Channel	MQZAO_CREATE **(2 on page 561)**
Client connection channel	MQZAO_CREATE **(2 on page 561)**
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Create *object* **(with replace) (3 on page 561, 4 on page 561)**

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Delete *object*

Object	Authorization required
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	MQZAO_DELETE
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE

Inquire *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY

Inquire *object* names

Object	Authorization required
Queue	No check
Topic	No check
Process	No check
Queue manager	No check
Namelist	No check
Authentication information	No check
Channel	No check
Client connection channel	No check
Listener	No check
Service	No check

Ping Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Reset Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Reset Queue Statistics

Object	Authorization required
Queue	MQZAO_DISPLAY and MQZAO_CHANGE
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	
Service	

Resolve Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Start Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Stop Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Note:

1. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the **GRTMQMAUT** command.
3. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
4. This option applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

Generic OAM profiles on IBM i

Object authority manager (OAM) generic profiles enable you to set the authority a user has to many objects at once, rather than having to issue separate **GRTMQMAUT** commands against each individual object when it is created. Using generic profiles in the **GRTMQMAUT** command enables you to set a generic authority for all future objects created that fit that profile.

The rest of this section describes the use of generic profiles in more detail:
- "Using wildcard characters"
- "Profile priorities"

Using wildcard characters

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the question mark (?) wildcard character matches any single character in a name. So, if you specify ABC.?EF, the authorization you give to that profile applies to any objects created with the names ABC.DEF, ABC.CEF, ABC.BEF, and so on.

The wildcard characters available are:

? Use the question mark (?) instead of any single character. For example, AB.?D would apply to the objects AB.CD, AB.ED, and AB.FD.

* Use the asterisk (*) as:
- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in ABC.DEF.GHI, the qualifiers are ABC, DEF, and GHI.

  For example, ABC.*.JKL would apply to the objects ABC.DEF.JKL, and ABC.GHI.JKL. (Note that it would **not** apply to ABC.JKL ; * used in this context always indicates one qualifier.)
- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, ABC.DE*.JKL would apply to the objects ABC.DE.JKL, ABC.DEF.JKL, and ABC.DEGH.JKL.

** Use the double asterisk (**) ***once*** in a profile name as:
- The entire profile name to match all object names. For example, if you use the keyword OBJTYPE (*PRC) to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, **.ABC identifies all objects with the final qualifier ABC.

Profile priorities

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
GRTMQMAUT OBJ(AB.*) OBJTYPE(*Q) USER(FRED) AUT(*PUT) MQMNAME(MYQMGR)
GRTMQMAUT OBJ(AB.C*) OBJTYPE(*Q) USER(FRED) AUT(*GET) MQMNAME(MYQMGR)
```

The first gives put authority to all queues for the principal FRED with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either GRTMQMAUT could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific than a generic character. So, in the previous example, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:
1. ?
2. *
3. **

Specifying the installed authorization service on IBM i

You can specify which authorization service component to use.

The parameter **`Service Component name`** on **`GRTMQMAUT`** and **`RVKMQMAUT`** allows you to specify the name of the installed authorization service component.

Selecting **F24** on the initial panel, followed by **F9=All parameters** on the next panel of either command, allows you to specify either the installed authorization component (*DFT) or the name of the required authorization service component specified in the Service stanza of the queue manager's qm.ini file.

`DSPMQMAUT` also has this extra parameter. This parameter allows you to search all the installed authorization components (*DFT), or the specified authorization-service component name, for the specified object name, object type, and user

Working with and without authority profiles on IBM i

Use this information to learn how to work with authority profiles and how to work without authority profiles.

You can work with authority profiles, as explained in "Working with authority profiles," or without them, as explained here:

To work without authority profiles, use `*NONE` as an Authority parameter on **`GRTMQMAUT`** to create profiles without authority. This leaves any existing profiles unchanged.

On **`RVKMQMAUT`**, use `*REMOVE` as an Authority parameter to remove an existing authority profile.

Working with authority profiles

There are two commands associated with authority profiling:
- **`WRKMQMAUT`**
- **`WRKMQMAUTD`**

You can access these commands directly from the command line, or from the WRKMQM panel by:
1. Typing in the queue manager name and pressing the `Enter` key to access the **`WRKMQM`** results panel.
2. Selecting `F23=More options` on this panel.

Option 24 selects the results panel for the **`WRKMQMAUT`** command and option 25 selects the **`WRKMQMAUTI`** command, which is used with the SSL bindings layer.

WRKMQMAUT

This command allows you to work with the authority data held in the authority queue.

Note: To run this command you must have `*connect` and `*admdsp` authority to the queue manager. However, to create or delete a profile, you need QMQMADM authority.

If you output the information to the screen, a list of authority profile names, together with their types, is displayed. If you print the output, you receive a detailed list of all the authority data, the registered users, and their authorities.

Entering an object or profile name on this panel, and pressing ENTER takes you to the results panel for **`WRKMQMAUT`** .

If you select `4=Delete`, you go to a new panel from which you can confirm that you want to delete all the user names registered to the generic authority profile name you specify. This option runs **`RVKMQMAUT`** with the option *REMOVE for all the users, and applies **only** to generic profile names.

If you select `12=Work with profile` you go to the **`WRKMQMAUTD`** command results panel, as explained in "WRKMQMAUTD."

WRKMQMAUTD

This command allows you to display all the users registered with a particular authority profile name and object type. To run this command you must have `*connect` and `*admdsp` authority to the queue manager. However, to grant, run, create, or delete a profile you need QMQMADM authority.

Selecting `F24=More keys` from the initial input panel, followed by option `F9=All Parameters` displays the Service Component Name as for **`GRTMQMAUT`** and **`RVKMQMAUT`**.

Note: The `F11=Display Object Authorizations` key toggles between the following types of authorities:
- Object authorizations
- Context authorizations
- MQI authorizations

The options on the screen are:

2=Grant
Takes you to the **`GRTMQMAUT`** panel to add to the current authorities.

3=Revoke
Takes you to the **`RVKMQMAUT`** panel to remove some of the current definitions

4=Delete
Takes you to a panel that allows you to delete the authority data for specified users. This runs **`RVKMQMAUT`** with the option *REMOVE.

5=Display
Takes you to the existing **`DSPMQMAUT`** command

F6=Create
Takes you to the **`GRTMQMAUT`** panel that allows you to create a profile authority record.

Object authority manager guidelines for IBM i

Additional hints and tips for using the object authority manager (OAM)

Limit access to sensitive operations

Some operations are sensitive; limit them to privileged users. For example,

- Accessing some special queues, such as transmission queues or the command queue `SYSTEM.ADMIN.COMMAND.QUEUE`
- Running programs that use full MQI context options
- Creating and copying application queues

Queue manager directories

The directories and libraries containing queues and other queue manager data are private to the product. Do not use standard operating system commands to grant or revoke authorizations to MQI resources.

Queues

The authority to a dynamic queue is based on, but is not necessarily the same as, that of the model queue from which it is derived.

For alias queues and remote queues, the authorization is that of the object itself, not the queue to which the alias or remote queue resolves. It is possible to authorize a user profile to access an alias queue that resolves to a local queue to which the user profile has no access permissions.

Limit the authority to create queues to privileged users. If you do not, users can bypass the normal access control by creating an alias.

Alternate-user authority

Alternate-user authority controls whether one user profile can use the authority of another user profile when accessing an IBM MQ object. This technique is essential where a server receives requests from a program and the server wants to ensure that the program has the required authority for the request. The server might have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example:

- A server program running under user profile PAYSERV retrieves a request message from a queue that was put on the queue by user profile USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.
- Instead of using its own user profile (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user profile, in this case, USER1. In this example, you can use alternate-user authority to control whether PAYSERV is allowed to specify USER1 as an alternate-user profile when it opens the reply-to queue.

The alternate-user profile is specified on the *`AlternateUserId`* field of the object descriptor.

Note: You can use alternate-user profiles on any IBM MQ object. Use of an alternate-user profile does not affect the user profile used by any other resource managers.

Context authority

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message.

For descriptions of the message descriptor fields relating to context, see MQMD overview.

For information about the context options, see Message context.

Remote security considerations

For remote security, consider:

Put authority
: For security across queue managers, you can specify the put authority that is used when a channel receives a message sent from another queue manager.

  This parameter is valid only for RCVR, RQSTR, or CLUSRCVR channel types. Specify the channel attribute PUTAUT as follows:

  DEF Default user profile. This is the QMQM user profile under which the message channel agent is running.

  CTX The user profile in the message context.

Transmission queues
: Queue managers automatically put remote messages on a transmission queue; no special authority is required. However, putting a message directly on a transmission queue requires special authorization.

Channel exits
: Channel exits can be used for added security.

Channel authentication records
: Use to exercise more precise control over the access granted to connecting systems at a channel level.

For more information about remote security, see "Channel authorization" on page 485.

Protecting channels with SSL or TLS

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols provide channel security, with protection against eavesdropping, tampering, and impersonation. IBM MQ support for SSL and TLS enables you to specify, on the channel definition, that a particular channel uses SSL or TLS security. You can also specify details of the security you want, such as the encryption algorithm you want to use.

SSL and TLS support in IBM MQ uses the queue manager *authentication information object* and various CL and MQSC commands and queue manager and channel parameters that define the SSL or TLS support required in detail.

The following CL commands support SSL or TLS:

WRKMQMAUTI
: Work with the attributes of an authentication information object.

CHGMQMAUTI
: Modify the attributes of an authentication information object.

CRTMQMAUTI
: Create an authentication information object.

CPYMQMAUTI
Create an authentication information object by copying an existing one.

DLTMQMAUTI
Delete an authentication information object.

DSPMQMAUTI
Displays the attributes for a specific authentication information object.

For an overview of channel security using SSL or TLS, see
- Protecting channels with SSL

For details of PCF commands associated with SSL, see
- Change, Copy, and Create Authentication Information Object
- Delete Authentication Information Object
- Inquire Authentication Information Object

Setting up security on z/OS

Security considerations specific to z/OS.

Security in IBM MQ for z/OS is controlled using RACF or an equivalent external security manager (ESM).

If a userid has `uid0`, it has access to the entire file system, that is, Superuser access. Under RACF, the only way to restrict this is by using the feature FSACCESS, which can restrict access at the file system level by RACF userid. For more information, see Using the FSACCESS class profile to restrict access in the *z/OS UNIX System Services Planning* documentation.

The following instructions assume that you are using RACF.

Related information:
Security scenario: two queue managers on z/OS
Security scenario: queue-sharing group on z/OS

RACF security classes

RACF classes are used to hold the profiles required for IBM MQ security checking. Many of the member classes have equivalent group classes. You must activate the classes and enable them to accept generic profiles

Each RACF class holds one or more profiles used at some point in the checking sequence, as shown in Table 35.

Table 35. RACF classes used by IBM MQ

Member class	Group class	Contents
MQADMIN	GMQADMIN	Profiles: Used mainly for holding profiles for administration-type functions. For example: • Profiles for IBM MQ security switches • The RESLEVEL security profile • Profiles for alternate user security • The context security profile • Profiles for command resource security

Table 35. RACF classes used by IBM MQ (continued)

Member class	Group class	Contents
MXADMIN	GMXADMIN	Profiles: Used mainly for holding profiles for administration-type functions. For example: • Profiles for IBM MQ security switches • The RESLEVEL security profile • Profiles for alternate user security • The context security profile • Profiles for command resource security This class can hold both uppercase and mixed case RACF profiles.
MQCONN		Profiles used for connection security
MQCMDS		Profiles used for command security
MQQUEUE	GMQQUEUE	Profiles used in queue resource security
MXQUEUE	GMXQUEUE	Mixed case and uppercase profiles used in queue resource security
MQPROC	GMQPROC	Profiles used in process resource security
MXPROC	GMXPROC	Mixed case and uppercase profiles used in process resource security
MQNLIST	GMQNLIST	Profiles used in namelist resource security
MXNLIST	GMXNLIST	Mixed case and uppercase profiles used in namelist resource security
MXTOPIC	GMXTOPIC	Mixed case and uppercase profiles used in topic security

Some classes have a related *group class* that enables you to put together groups of resources that have similar access requirements. For details about the difference between the member and group classes and when to use a member or group class, see the *z/OS SecureWay Security Server RACF Security Administrator's Guide*.

The classes must be activated before security checks can be made. To activate all the IBM MQ classes, you can use this RACF command:

```
SETROPTS CLASSACT(MQADMIN,MXADMIN,MQQUEUE,MXQUEUE,MQPROC,MXPROC,
                 MQNLIST,MXNLIST,MXTOPIC,MQCONN,MQCMDS)
```

You should also ensure that you set up the classes so that they can accept generic profiles. You also do this with the RACF command SETROPTS, for example:

```
SETROPTS GENERIC(MQADMIN,MXADMIN,MQQUEUE,MXQUEUE,MQPROC,MXPROC,
                MQNLIST,MXNLIST,MXTOPIC,MQCONN,MQCMDS)
```

RACF profiles

All RACF profiles used by IBM MQ contain a prefix, which is either the queue manager name or the queue-sharing group name. Be careful when you use the percent sign as a wildcard.

All RACF profiles used by IBM MQ contain a prefix. For queue-sharing group level security, this is the queue-sharing group name. For queue manager level security, the prefix is the queue manager name. If you are using a mixture of queue manager and queue-sharing group level security, you will use profiles with both types of prefix. (Queue-sharing group and queue manager level security are described in IBM MQ for z/OS concepts: security.)

For example, if you want to protect a queue called QUEUE_FOR_SUBSCRIBER_LIST in queue-sharing group QSG1 at queue-sharing group level, the appropriate profile would be defined to RACF as:

```
RDEFINE MQQUEUE QSG1.QUEUE_FOR_SUBSCRIBER_LIST
```

If you want to protect a queue called QUEUE_FOR_LOST_CARD_LIST, that belongs to queue manager STCD at queue manager level, the appropriate profile would be defined to RACF as:

```
RDEFINE MQQUEUE STCD.QUEUE_FOR_LOST_CARD_LIST
```

This means that different queue managers and queue-sharing groups can share the same RACF database and yet have different security options.

Do not use generic queue manager names in profiles to avoid unanticipated user access.

IBM MQ allows the use of the percent sign (%) in object names. However, RACF uses the % character as a single-character wildcard. This means that when you define an object name with a % character in its name, you must consider this when you define the corresponding profile.

For example, for the queue CREDIT_CARD_%_RATE_INQUIRY, on queue manager CRDP, the profile would be defined to RACF as follows:

```
RDEFINE MQQUEUE CRDP.CREDIT_CARD_%_RATE_INQUIRY
```

This queue cannot be protected by a generic profile, such as, CRDP.**.

IBM MQ allows the use of mixed case characters in object names. You can protect these objects by defining:

1. Mixed case profiles in the appropriate mixed case RACF classes, or
2. Generic profiles in the appropriate uppercase RACF classes.

To use mixed case profiles and mixed case RACF classes you must follow the steps described in z/OS: Migrating a queue manager to mixed case security.

There are some profiles, or parts of profiles, that remain uppercase only as the values are provided by IBM MQ. These are:

- Switch profiles.
- All high-level qualifiers (HLQ) including subsystem and Queue-Sharing Group identifiers.
- Profiles for SYSTEM objects.
- Profiles for Default objects.
- The **MQCMDS** class, so all command profiles are uppercase only.
- The **MQCONN** class, so all connection profiles are uppercase only.
- **RESLEVEL** profiles.

- The `'object'` qualification in command resource profiles; for example, `hlq.QUEUE.queuename`. The resource name only is mixed case.
- Dynamic queue profiles `hlq.CSQOREXX.*`, `hlq.CSQUTIL.*`, and `CSQXCMD.*`.
- The `'CONTEXT '` part of `hlq.CONTEXT.resourcename`.
- The `'ALTERNATE.USER'` part of `hlq.ALTERNATE.USER.userid`.

For example, if you have a queue called `PAYROLL.Dept1` on Queue Manager `QM01` and you are using:
- Mixed case profiles; you can define a profile in the IBM MQ RACF class `MXQUEUE`
  ```
  RDEFINE MXQUEUE MQ01.PAYROLL.Dept1
  ```
- Uppercase profiles; you can define a profile in the IBM MQ RACF class `MQQUEUE`
  ```
  RDEFINE MQQUEUE MQ01.PAYROLL.*
  ```

The first example, using mixed case profiles, gives you more granular control over granting authority to access the resource.

Switch profiles

To control the security checking performed by IBM MQ, you use *switch profiles*. A switch profile is a normal RACF profile that has a special meaning to IBM MQ. The access list in switch profiles is not used by IBM MQ.

IBM MQ maintains an internal switch for each switch type shown in tables Switch profiles for subsystem level security, Switch profiles for queue-sharing group or queue manager level security,and Switch profiles for resource checking. Switch profiles can be maintained at queue-sharing group level, or at queue manager level, or at a combination of both. Using a single set of queue-sharing group security switch profiles, you can control security on all the queue managers within a queue-sharing group.

When a security switch is set on, the security checks associated with the switch are performed. When a security switch is set off, the security checks associated with the switch are bypassed. The default is that all security switches are set on.

Switches and classes

When you start a queue manager or refresh security, IBM MQ sets switches according to the state of various RACF classes.

When a queue manager is started (or when the MQADMIN or MXADMIN class is refreshed by the IBM MQ REFRESH SECURITY command), IBM MQ first checks the status of RACF and the appropriate class:
- The MQADMIN class if you are using uppercase profiles
- The MXADMIN class if you are using mixed case profile.

It sets the subsystem security switch off if any of these conditions is true:
- RACF is inactive or not installed.
- The MQADMIN or MXADMIN class is not defined (these classes are always defined for RACF because they are included in the class descriptor table (CDT)).
- The MQADMIN or MXADMIN class has not been activated.

If both RACF and the MQADMIN or MXADMIN class are active, IBM MQ checks the MQADMIN or MXADMIN class to see whether any of the switch profiles have been defined. It first checks the profiles described in "Profiles to control subsystem security" on page 571. If subsystem security is not required, IBM MQ sets the internal subsystem security switch off, and performs no further checks.

The profiles determine whether the corresponding IBM MQ switch is set on or off.
- If the switch is off, that type of security is deactivated.

- If any IBM MQ switch is set on, IBM MQ checks the status of the RACF class associated with the type of security corresponding to the IBM MQ switch. If the class is not installed or not active, the IBM MQ switch is set off. For example, process security checks are not carried out if the MQPROC or MXPROC class has not been activated. The class not being active is equivalent to defining NO.PROCESS.CHECKS profile for every queue manager and queue-sharing group that uses this RACF database.

How switches work

To set off a security switch, define a NO.* switch profile for it. You can override a NO.* profile set at the queue-sharing group level by defining a YES.* profile for a queue manager.

To set off a security switch, you need to define a NO.* switch profile for it. The existence of a NO.* profile means that security checks are **not** performed for that type of resource, unless you choose to override a queue-sharing group level setting on a particular queue manager. This is described in "Overriding queue-sharing group level settings."

If your queue manager is not a member of a queue-sharing group, you do not need to define any queue-sharing group level profiles or any override profiles. However, you must remember to define these profiles if the queue manager joins a queue-sharing group at a later date.

Each NO.* switch profile that IBM MQ detects turns off the checking for that type of resource. Switch profiles are activated during startup of the queue manager. If you change the switch profiles while any affected queue managers are running, you can get IBM MQ to recognize the changes by issuing the IBM MQ REFRESH SECURITY command.

The switch profiles must always be defined in the MQADMIN or MXADMIN class. Do not define them in the GMQADMIN or GMXADMIN class. Tables Switch profiles for subsystem level security and Switch profiles for resource checking show the valid switch profiles and the security type they control.

Overriding queue-sharing group level settings

You can override queue-sharing group level security settings for a particular queue manager that is a member of that group. If you want to perform queue manager checks on an individual queue manager that are not performed on other queue managers in the group, use the (qmgr-name.YES.*) switch profiles.

Conversely, if you do not want to perform a certain check on one particular queue manager within a queue-sharing group, define a (qmgr-name.NO.*) profile for that particular resource type on the queue manager, and do not define a profile for the queue-sharing group. (IBM MQ only checks for a queue-sharing group level profile if it does not find a queue manager level profile.)

Profiles to control subsystem security

IBM MQ checks whether subsystem security checks are required for the subsystem, for the queue manager, and for the queue-sharing group.

The first security check made by IBM MQ is used to determine whether security checks are required for the whole IBM MQ subsystem. If you specify that you do not want subsystem security, no further checks are made.

The following switch profiles are checked to determine whether subsystem security is required. Figure 67 on page 572 shows the order in which they are checked.

Table 36. Switch profiles for subsystem level security

Switch profile name	Type of resource or checking that is controlled
qmgr-name.NO.SUBSYS.SECURITY	Subsystem security for this queue manager
qsg-name.NO.SUBSYS.SECURITY	Subsystem security for this queue-sharing group
qmgr-name.YES.SUBSYS.SECURITY	Subsystem security override for this queue manager

If your queue manager is not a member of a queue-sharing group, IBM MQ checks for the qmgr-name.NO.SUBSYS.SECURITY switch profile only.

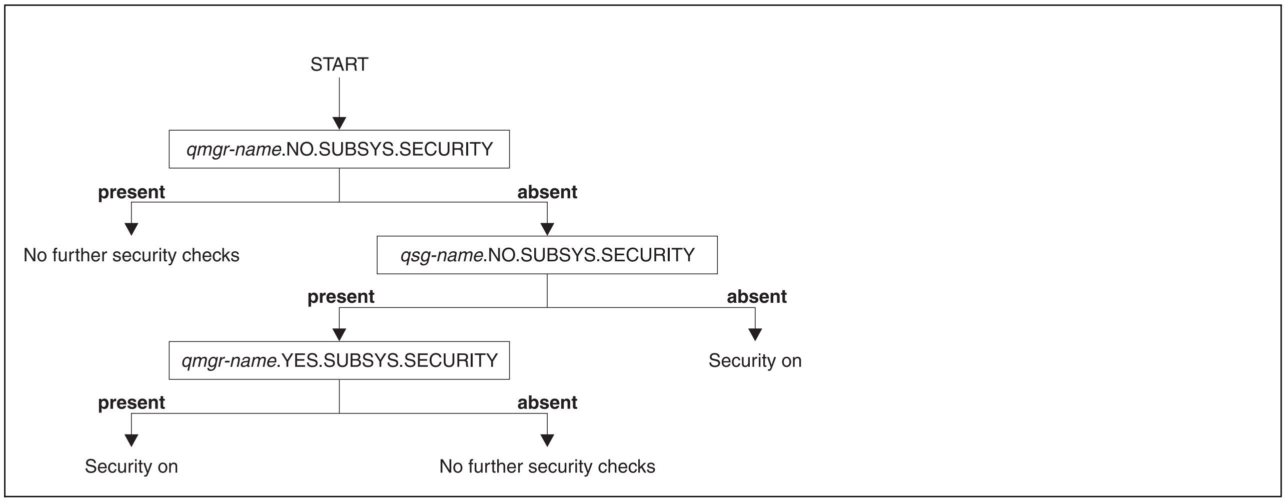

START
qmgr-name.NO.SUBSYS.SECURITY
present
absent
No further security checks
qsg-name.NO.SUBSYS.SECURITY
present
absent
qmgr-name.YES.SUBSYS.SECURITY
Security on
present
absent
Security on
No further security checks

Figure 67. Checking for subsystem security

Profiles to control queue-sharing group or queue manager level security

If subsystem security checking is required, IBM MQ checks whether security checking is required at queue-sharing group or queue manager level.

When IBM MQ has determined that security checking is required, it then determines whether checking is required at queue-sharing group or queue manager level, or both. These checks are not performed if your queue manager is not a member of a queue sharing group.

The following switch profiles are checked to determine the level required. Figure 68 on page 573 and Figure 69 on page 573 show the order in which they are checked.

Table 37. Switch profiles for queue-sharing group or queue manager level security

Switch profile name	Type of resource or checking that is controlled
qmgr-name.NO.QMGR.CHECKS	No queue manager level checks for this queue manager
qsg-name.NO.QMGR.CHECKS	No queue manager level checks for this queue-sharing group
qmgr-name.YES.QMGR.CHECKS	Queue manager level checks override for this queue manager
qmgr-name.NO.QSG.CHECKS	No queue-sharing group level checks for this queue manager
qsg-name.NO.QSG.CHECKS	No queue-sharing group level checks for this queue-sharing group
qmgr-name.YES.QSG.CHECKS	Queue-sharing group level checks override for this queue manager

If subsystem security is active, you cannot switch off both queue-sharing group and queue manager level security. If you try to do so, IBM MQ sets security checking on at both levels.

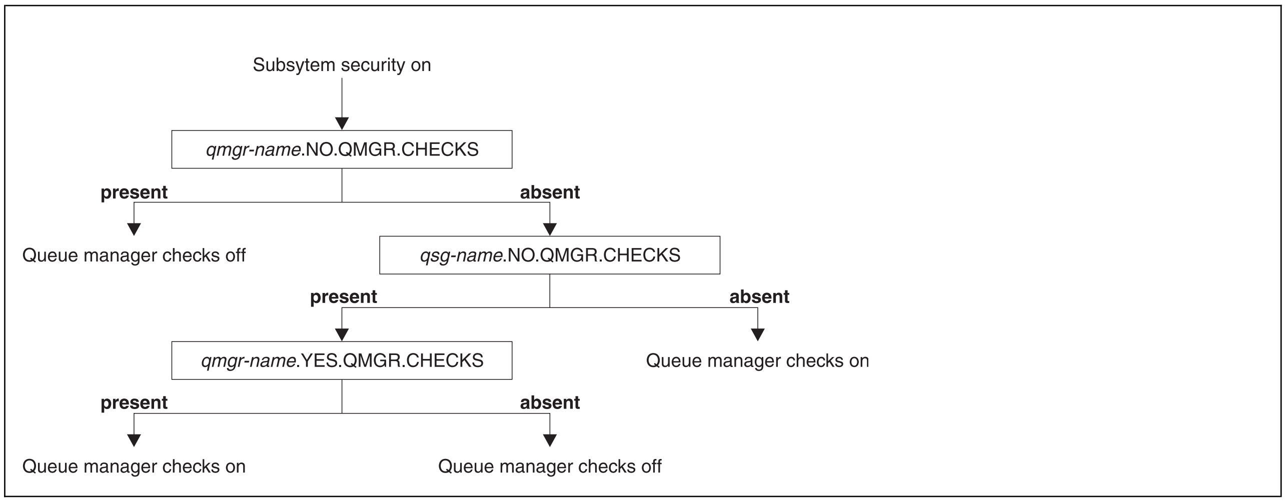


Figure 68. Checking for queue manager level security

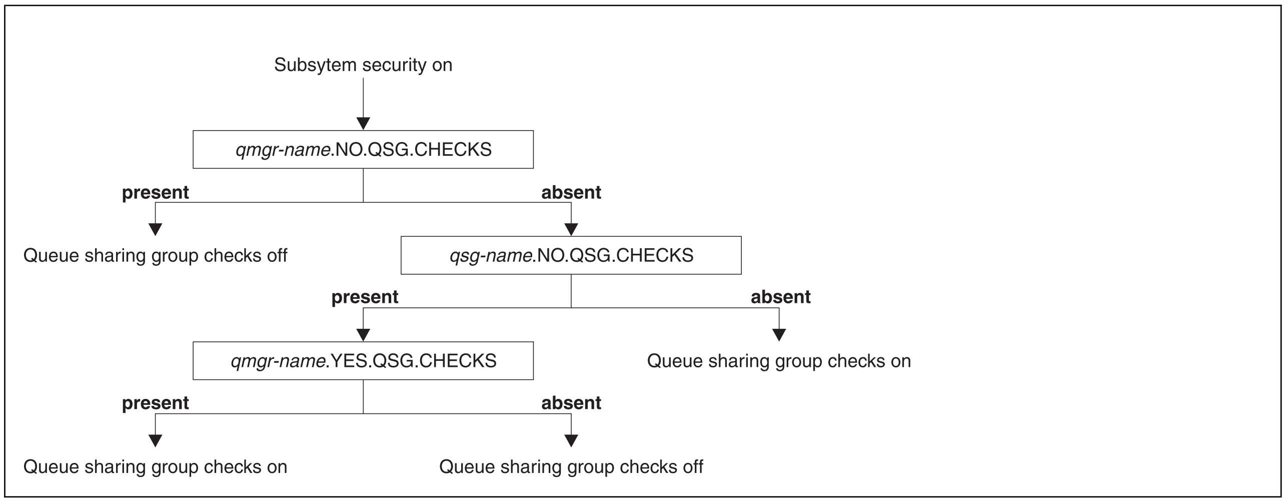


Figure 69. Checking for queue-sharing group level security

Valid combinations of security switches:

Only certain combinations of switches are valid. If you use a combination of switch settings that is not valid, message CSQH026I is issued and security checking is set on at both queue-sharing group and queue manager level.

Table 38, Table 39 on page 574, Table 40 on page 574, and Table 41 on page 574 show the sets of combinations of switch settings that are valid for each type of security level.

Table 38. Valid security switch combinations for queue manager level security

Combinations
qmgr-name.NO.QSG.CHECKS
qsg-name.NO.QSG.CHECKS
qmgr-name.NO.QSG.CHECKS qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS

Table 38. Valid security switch combinations for queue manager level security (continued)

Combinations
qsg-name.NO.QSG.CHECKS qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS

Table 39. Valid security switch combinations for queue-sharing group level security

Combinations
qmgr-name.NO.QMGR.CHECKS
qsg-name.NO.QMGR.CHECKS
qmgr-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS

Table 40. Valid security switch combinations for queue manager and queue-sharing group level security

Combinations
qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS No QSG.* profiles defined
No QMGR.* profiles defined qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS
No profiles for either switch defined

Table 41. Other valid security switch combinations that switch both levels of checking **on**.

Combinations
qmgr-name.NO.QMGR.CHECKS qmgr-name.NO.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS
qmgr-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qmgr-name.NO.QSG.CHECKS

Resource level checks

A number of switch profiles are used to control access to resources. Some stop checking being performed on either a queue manager or a queue-sharing group. These can be overridden by profiles that enable checking for specific queue managers.

Table 42 shows the switch profiles used to control access to IBM MQ resources.

If your queue manager is part of a queue sharing group and you have both queue manager and queue-sharing group security active, you can use a YES.* switch profile to override queue-sharing group level profiles and specifically turn on security for a particular queue manager.

Some profiles apply to both queue managers and queue-sharing groups. These are prefixed by the string *hlq* and you should substitute the name of your queue-sharing group or queue manager, as applicable. Profile names shown prefixed by *qmgr-name* are queue-manager override profiles; you should substitute the name of your queue manager.

Table 42. Switch profiles for resource checking

Type of resource checking that is controlled	Switch profile name	Override profile for a particular queue manager
Connection security	hlq.NO.CONNECT.CHECKS	qmgr-name.YES.CONNECT.CHECKS
Queue security	hlq.NO.QUEUE.CHECKS	qmgr-name.YES.QUEUE.CHECKS
Process security	hlq.NO.PROCESS.CHECKS	qmgr-name.YES.PROCESS.CHECKS
Namelist security	hlq.NO.NLIST.CHECKS	qmgr-name.YES.NLIST.CHECKS
Context security	hlq.NO.CONTEXT.CHECKS	qmgr-name.YES.CONTEXT.CHECKS
Alternate user security	hlq.NO.ALTERNATE.USER.CHECKS	qmgr-name.YES.ALTERNATE.USER.CHECKS
Command security	hlq.NO.CMD.CHECKS	qmgr-name.YES.CMD.CHECKS
Command resource security	hlq.NO.CMD.RESC.CHECKS	qmgr-name.YES.CMD.RESC.CHECKS
Topic security	hlq.NO.TOPIC.CHECKS	qmgr-name.YES.TOPIC.CHECKS
Note: Generic switch profiles such as hlq.NO.** are ignored by IBM MQ		

For example, if you want to perform process security checks on queue manager QM01, which is a member of queue-sharing group QSG3 but you do not want to perform process security checks on any of the other queue managers in the group, define the following switch profiles:

```
QSG3.NO.PROCESS.CHECKS
QM01.YES.PROCESS.CHECKS
```

If you want to have queue security checks performed on all the queue managers in the queue-sharing group, except QM02, define the following switch profile:

```
QM02.NO.QUEUE.CHECKS
```

(There is no need to define a profile for the queue sharing group because the checks are automatically enabled if there is no profile defined.)

An example of defining switches

Different IBM MQ subsystems have different security requirements, which can be implemented using different switch profiles.

Four IBM MQ subsystems have been defined:

- MQP1 (a production system)
- MQP2 (a production system)
- MQD1 (a development system)
- MQT1 (a test system)

All four queue managers are members of queue-sharing group QS01. All IBM MQ RACF classes have been defined and activated.

These subsystems have different security requirements:

- The production systems require full IBM MQ security checking to be active at queue-sharing group level on both systems.

  This is done by specifying the following profile:

  ```
  RDEFINE MQADMIN QS01.NO.QMGR.CHECKS
  ```

  This sets queue-sharing group level checking for all the queue managers in the queue-sharing group. You do not need to define any other switch profiles for the production queue managers because you want to check everything for these systems.
- Test queue manager MQT1 also requires full security checking. However, because you might want to change this later, security can be defined at queue-manager level so that you can change the security settings for this queue manager without affecting the other members of the queue-sharing group.

  This is done by defining the NO.QSG.CHECKS profile for MQT1 as follows:

  ```
  RDEFINE MQADMIN MQT1.NO.QSG.CHECKS
  ```
- Development queue manager MQD1 has different security requirements from the rest of the queue-sharing group. It requires only connection and queue security to be active.

  This is done by defining a `MQD1.YES.QMGR.CHECKS` profile for this queue manager, and then defining the following profiles to switch off security checking for the resources that do not need to be checked:

  ```
  RDEFINE MQADMIN MQD1.NO.CMD.CHECKS
  RDEFINE MQADMIN MQD1.NO.CMD.RESC.CHECKS
  RDEFINE MQADMIN MQD1.NO.PROCESS.CHECKS
  RDEFINE MQADMIN MQD1.NO.NLIST.CHECKS
  RDEFINE MQADMIN MQD1.NO.CONTEXT.CHECKS
  RDEFINE MQADMIN MQD1.NO.ALTERNATE.USER.CHECKS
  ```

When the queue manager is active, you can display the current security settings by issuing the DISPLAY SECURITY MQSC command.

You can also change the switch settings when the queue manager is running by defining or deleting the appropriate switch profile in the MQADMIN class. To make the changes to the switch settings active, you must issue the REFRESH SECURITY command for the MQADMIN class.

See "Refreshing queue manager security on z/OS" on page 628 for more details about using the DISPLAY SECURITY and REFRESH SECURITY commands.

Profiles used to control access to IBM MQ resources

You must define RACF profiles to control access to IBM MQ resources, in addition to the switch profiles that might have been defined. This collection of topics contains information about the RACF profiles for the different types of IBM MQ resource.

If you do not have a resource profile defined for a particular security check, and a user issues a request that would involve making that check, IBM MQ denies access. You do not have to define profiles for security types relating to any security switches that you have deactivated.

Profiles for connection security

If connection security is active, you must define profiles in the MQCONN class and permit the necessary groups or user IDs access to those profiles, so that they can connect to IBM MQ.

To enable a connection to be made, you must grant users RACF READ access to the appropriate profile. (If no queue manager level profile exists, and your queue manager is a member of a queue-sharing group, checks might be made against queue-sharing group level profiles, if the security is set up to do this.)

A connection profile qualified with a queue manager name controls access to a specific queue manager and users given access to this profile can connect to that queue manager. A connection profile qualified with queue-sharing group name controls access to all queue managers within the queue-sharing group for that connection type. For example, a user with access to `QS01.BATCH` can use a batch connection to any queue manager in queue-sharing group QS01 that has not got a queue manager level profile defined.

Note:

1. For information about the user IDs checked for different security requests, see "User IDs for security checking" on page 617.
2. Resource level security (RESLEVEL) checks are also made at connection time. For details, see "The RESLEVEL security profile" on page 610.

IBM MQ security recognizes the following different types of connection:

- Batch (and batch-type) connections, these include:
  - z/OS batch jobs
  - TSO applications
  - USS sign-ons
  - Db2 stored procedures
- CICS connections
- IMS connections from control and application processing regions
- The IBM MQ channel initiator

Connection security profiles for batch connections:

Profiles for checking batch-type connections are composed of the queue manager or queue-sharing group name followed by the word *BATCH*. Give the user ID associated with the connecting address space READ access to the connection profile.

Profiles for checking batch and batch-type connections take the form:

```
hlq.BATCH
```

where `hlq` can be either the `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name). If you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue-sharing group name. If it fails to find either profile, the connection request fails.

For batch or batch-type connection requests, you must permit the user ID associated with the connecting address space to access the connection profile. For example, the following RACF command allows users in the CONNTQM1 group to connect to the queue manager TQM1; these user IDs will be permitted to use any batch or batch-type connection.

```
RDEFINE MQCONN TQM1.BATCH UACC(NONE)
PERMIT TQM1.BATCH CLASS(MQCONN) ID(CONNTQM1) ACCESS(READ)
```

Using **`CHCKLOCL`** *on locally bound applications:*

`CHCKLOCL` only applies to connections that are made through BATCH connections and does not apply to connections made from CICS or IMS. Connections made through the channel initiator are controlled by **`CHCKCLNT`**.

Overview

If you want to configure your z/OS queue manager to mandate user ID and password checking for some, but not all, of your locally bound applications, you need to do some additional configuration.

The reason for this is that once **`CHCKLOCL`** (*REQUIRED*) is configured, legacy batch applications that use the MQCONN API call can no longer connect to the queue manager.

For z/OS only, a more granular mechanism based on the connection security of an address space can be used to downgrade the global CHCKLOCL(REQUIRED) configuration to CHCKLOCL(OPTIONAL) for specifically defined user IDs. The mechanism used, is described in the following text, together with an example.

In order to allow more granularity on **`CHCKLOCL`** (*REQUIRED*) than just EVERYONE, you modify **`CHCKLOCL`** in the same manner as you modify the access level of the user ID associated with the connecting address space to the `hlq.batch` connection profiles in the MQCONN class.

If the address space user ID only has READ access, which is the minimum you require to be able to connect at all, the **`CHCKLOCL`** configuration applies as written.

If the address space user ID has UPDATE access (or above) then the **`CHCKLOCL`** configuration operates in *OPTIONAL* mode. That is, you do not have to provide a user ID and password, but if you do, the user ID and password must be a valid pair.

Connection security already configured for your z/OS queue manager

If you have connection security configured for your z/OS queue manager and you want **`CHCKLOCL`** (*REQUIRED*) to apply to WAS locally bound applications, and no others, carry out the following steps:

1. Start with **CHCKLOCL** (*OPTIONAL*) as your configuration. This means that any user ID and passwords that are supplied are checked for validity, but not mandated.
2. List all the users that have access to the connection security profiles by issuing the command:
   ```
   RLIST MQCONN MQ23.BATCH AUTHUSER
   ```
   This command displays, for example:
   ```
   CLASS    NAME
   -----    ----
   MQCONN   MQ23.BATCH

   USER     ACCESS  ACCESS COUNT
   ----     ------  ------ -----
   JOHNDOE  READ    000009
   JDOE1    READ    000003
   WASUSER  READ    000000
   ```
3. For each user ID listed as having READ access, change the access to
   ```
   UPDATE:- PERMIT MQ23.BATCH CLASS(MQCONN) ID(JOHNDOE) ACCESS(UPDATE)
   ```
4. Update the IBM MQ configuration to **CHCKLOCL** (*REQUIRED*).

   The combination of UPDATE access to `MQ23.BATCH` and the current setting means that you are using **CHCKLOCL** (*OPTIONAL*).
5. Now, apply the **CHCKLOCL** (*REQUIRED*) behavior to one specific user ID, for example `WASUSER`, so that all the connections coming from that region must provide a user ID and password.
6. Do this by reversing the change you made previously, by issuing the command:
   ```
   PERMIT MQ23.BATCH CLASS(MQCONN) ID(WASUSER) ACCESS(READ)
   ```

Connection security is not configured for your z/OS queue manager

In this situation, you must:
1. Create connection profiles for `hlq.BATCH` in the MQCONN class, by issuing the command:
   ```
   RDEFINE MQCONN MQ23.BATCH UACC(NONE)
   ```
2. Authorize all user IDs that create batch connections to the queue manager, so that they have UPDATE access to this profile. Doing this bypasses the **CHCKLOCL** (*REQUIRED*) requirement for the user ID and password at the time of connection.
3. Do this by issuing the command:
   ```
   PERMIT MQ23.BATCH CLASS(MQCONN)ID(JOHNDOE) ACCESS(UPDATE)
   ```
   These include user IDs:
   a. Used for CSQUTIL, ISPF panels, and other locally bound tools.
   b. Associated with batch like connections to the queue manager. Consider for example, IBM MQ Advanced Message Security, IBM Integration Bus, Db2 stored procedures, USS and TSO users, and Java applications
4. Delete the switch profile for the queue manager by issuing the command:
   ```
   hlq.NO.CONNECT.CHECKS
   ```
5. Now, apply the **CHCKLOCL** (*REQUIRED*) behavior to one specific user ID, for example `WASUSER`, so that all the connections coming from that region must provide a user ID and password.
6. Do this by reversing the change you made previously, by issuing the command:
   ```
   PERMIT MQ23.BATCH CLASS(MQCONN) ID(WASUSER) ACCESS(READ)
   ```

Connection security profiles for CICS connections:

Profiles for checking CICS connections are composed of the queue manager or queue-sharing group name followed by the word *CICS* . Give the user ID associated with the CICS address space READ access to the connection profile.

Profiles for checking connections from CICS take the form:

```
hlq.CICS
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name). If you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue-sharing group name. If it fails to find either profile, the connection request fails

For connection requests by CICS, you need only permit the CICS address space user ID access to the connection profile.

For example, the following RACF commands allow the CICS address space user ID KCBCICS to connect to the queue manager TQM1:

```
RDEFINE MQCONN TQM1.CICS UACC(NONE)
PERMIT TQM1.CICS CLASS(MQCONN) ID(KCBCICS) ACCESS(READ)
```

Connection security profiles for IMS connections:

Profiles for checking IMS connections are composed of the queue manager or queue-sharing group name followed by the word *IMS* . Give the IMS control and dependent region user IDs READ access to the connection profile.

Profiles for checking connections from IMS take the form:

```
hlq.IMS
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name). If you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue-sharing group name. If it fails to find either profile, the connection request fails

For connection requests by IMS, permit access to the connection profile for the IMS control and dependent region user IDs.

For example, the following RACF commands allow:
- The IMS region user ID, IMSREG, to connect to the queue manager TQM1.
- Users in group BMPGRP to submit BMP jobs.

```
RDEFINE MQCONN TQM1.IMS UACC(NONE)
PERMIT TQM1.IMS CLASS(MQCONN) ID(IMSREG,BMPGRP) ACCESS(READ)
```

Connection security profiles for the channel initiator:

Profiles for checking connections from the channel initiator are composed of the queue manager or queue-sharing group name followed by the word *CHIN*. Give the user ID used by the channel initiator started task address space READ access to the connection profile.

Profiles for checking connections from the channel initiator take the form:

```
hlq.CHIN
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name). If you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue-sharing group name. If it fails to find either profile, the connection request fails

For connection requests by the channel initiator, define access to the connection profile for the user ID used by the channel initiator started task address space.

For example, the following RACF commands allow the channel initiator address space running with user ID DQCTRL to connect to the queue manager TQM1:

```
RDEFINE MQCONN TQM1.CHIN UACC(NONE)
PERMIT TQM1.CHIN CLASS(MQCONN) ID(DQCTRL) ACCESS(READ)
```

Profiles for queue security

If queue security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to these profiles. Queue security profiles are named after the queue manager or queue-sharing group, and the queue to be opened.

If queue security is active, you must:

- Define profiles in the **`MQQUEUE`** or **`GMQQUEUE`** classes if using uppercase profiles.
- Define profiles in the **`MXQUEUE`** or **`GMXQUEUE`** classes if using mixed case profiles.
- Permit the necessary groups or user IDs access to these profiles, so that they can issue IBM MQ API requests that use queues.

Profiles for queue security take the form:

```
hlq.queuename
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name), and `queuename` is the name of the queue being opened, as specified in the object descriptor on the **MQOPEN** or **MQPUT1** call.

A profile prefixed by the queue manager name controls access to a single queue on that queue manager. A profile prefixed by the queue-sharing group name controls access to access to one or more queues with that queue name on all queue managers within the queue-sharing group, or access to a shared queue by any queue manager within the group. This access can be overridden on an individual queue manager by defining a queue-manager level profile for that queue on that queue manager.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

If you are using shared queues, you are recommended to use queue-sharing group level security.

For details of how queue security operates when the queue name is that of an alias or a model queue z/OS, see "Considerations for alias queues" on page 584 and "Considerations for model queues" on page 585.

The RACF access required to open a queue depends on the **MQOPEN** or **MQPUT1** options specified. If more than one of the MQOO_* and MQPMO_* options is coded, the queue security check is performed for the highest RACF authority required.

Table 43. Access levels for queue security using the MQOPEN or MQPUT1 calls

MQOPEN or MQPUT1 option	RACF access level required to access hlq.queuename
MQOO_BROWSE	READ
MQOO_INQUIRE	READ
MQOO_BIND_*	UPDATE
MQOO_INPUT_*	UPDATE
MQOO_OUTPUT or MQPUT1	UPDATE
MQOO_PASS_ALL_CONTEXT MQPMO_PASS_ALL_CONTEXT	UPDATE
MQOO_PASS_IDENTITY_CONTEXT MQPMO_PASS_IDENTITY_CONTEXT	UPDATE
MQOO_SAVE_ALL_CONTEXT	UPDATE
MQOO_SET_IDENTITY_CONTEXT MQPMO_SET_IDENTITY_CONTEXT	UPDATE
MQOO_SET_ALL_CONTEXT MQPMO_SET_ALL_CONTEXT	UPDATE
MQOO_SET	ALTER

For example, on IBM MQ queue manager QM77, all user IDs in the RACF group PAYGRP are to be given access to get messages from or put messages to all queues with names beginning with 'PAY.'. You can do this using these RACF commands:

```
RDEFINE MQQUEUE QM77.PAY.** UACC(NONE)
PERMIT QM77.PAY.** CLASS(MQQUEUE) ID(PAYGRP) ACCESS(UPDATE)
```

Also, all user IDs in the PAYGRP group must have access to put messages on queues that do not follow the PAY naming convention. For example:

```
REQUEST_QUEUE_FOR_PAYROLL
SALARY.INCREASE.SERVER
REPLIES.FROM.SALARY.MODEL
```

You can do this by defining profiles for these queues in the GMQQUEUE class and giving access to that class as follows:

```
RDEFINE GMQQUEUE PAYROLL.EXTRAS UACC(NONE)
        ADDMEM(QM77.REQUEST_QUEUE_FOR_PAYROLL,
               QM77.SALARY.INCREASE.SERVER,
               QM77.REPLIES.FROM.SALARY.MODEL)
PERMIT PAYROLL.EXTRAS CLASS(GMQQUEUE) ID(PAYGRP) ACCESS(UPDATE)
```

Note:

1. If the RACF access level that an application has to a queue security profile is changed, the changes only take effect for any new object handles obtained (that is, new **MQOPEN** s) for that queue. Those handles already in existence at the time of the change retain their existing access to the queue. If an application is required to use its changed access level to the queue rather than its existing access level, it must close and reopen the queue for each object handle that requires the change.
2. In the example, the queue manager name `QM77` could also be the name of a queue-sharing group.

Other types of security checks might also occur at the time the queue is opened depending on the open options specified and the types of security that are active. z/OS See also "Profiles for context security" on page 598 and "Profiles for alternate user security" on page 596. For a summary table showing the open options and the security authorization needed when queue, context, and alternate user security are all active, see Table 48 on page 589.

If you are using publish/subscribe you must consider the following. When an MQSUB request is processed a security check is performed to ensure that the user ID making the request has the required access to put messages to the target IBM MQ queue as well as the required access to subscribe to the IBM MQ topic.

Table 44. Access levels for queue security using the MQSUB call

MQSUB option	RACF access level required to access hlq.queuename
MQSO_ALTER, MQSO_CREATE, and MQSO_RESUME	UPDATE

Note:

1. The `hlq.queuename` is the destination queue for publications. When this is a managed queue, you need access to the appropriate model queue to be used for the managed queue and the dynamic queue that are created.
2. You can use a technique like this for the destination queue you provide on an MQSUB API call if you want to distinguish between the users making the subscriptions, and the users retrieving the publications from the destination queue.

Considerations for alias queues:

When you issue an **MQOPEN** or **MQPUT1** call for an alias queue, IBM MQ makes a resource check against the queue name specified in the object descriptor (MQOD) on the call. It does not check if the user is allowed access to the target queue name.

For example, an alias queue called PAYROLL.REQUEST resolves to a target queue of PAY.REQUEST. If queue security is active, you need only be authorized to access the queue PAYROLL.REQUEST. No check is made to see if you are authorized to access the queue PAY.REQUEST.

Using alias queues to distinguish between MQGET and MQPUT requests:

The range of MQI calls available in one access level can cause a problem if you want to restrict access to a queue to allow only the **MQPUT** call or only the **MQGET** call. A queue can be protected by defining two aliases that resolve to that queue: one that enables applications to get messages from the queue, and one that enable applications to put messages on the queue.

The following text gives you an example of how you can define your queues to IBM MQ:

```
DEFINE QLOCAL(MUST_USE_ALIAS_TO_ACCESS) GET(ENABLED)
       PUT(ENABLED)

DEFINE QALIAS(USE_THIS_ONE_FOR_GETS) GET(ENABLED)
       PUT(DISABLED) TARGQ(MUST_USE_ALIAS_TO_ACCESS)

DEFINE QALIAS(USE_THIS_ONE_FOR_PUTS) GET(DISABLED)
       PUT(ENABLED) TARGQ(MUST_USE_ALIAS_TO_ACCESS)
```

You must also make the following RACF definitions:

```
RDEFINE MQQUEUE hlq.MUST_USE_ALIAS_TO_ACCESS UACC(NONE)
RDEFINE MQQUEUE hlq.USE_THIS_ONE_FOR_GETS UACC(NONE)
RDEFINE MQQUEUE hlq.USE_THIS_ONE_FOR_PUTS UACC(NONE)
```

Then you ensure that no users have access to the queue hlq.MUST_USE_ALIAS_TO_ACCESS, and give the appropriate users or groups access to the alias. You can do this using the following RACF commands:

```
PERMIT hlq.USE_THIS_ONE_FOR_GETS CLASS(MQQUEUE)
       ID(GETUSER,GETGRP) ACCESS(UPDATE)
PERMIT hlq.USE_THIS_ONE_FOR_PUTS CLASS(MQQUEUE)
       ID(PUTUSER,PUTGRP) ACCESS(UPDATE)
```

This means user ID GETUSER and user IDs in the group GETGRP are only allowed to get messages on MUST_USE_ALIAS_TO_ACCESS through the alias queue USE_THIS_ONE_FOR_GETS; and user ID PUTUSER and user IDs in the group PUTGRP are only allowed to put messages through the alias queue USE_THIS_ONE_FOR_PUTS.

Note:

1. If you want to use a technique like this, you must inform your application developers, so that they can design their programs appropriately.
2. You can use a technique like this for the destination queue you provide on and MQSUB API request if you want to distinguish between the users making the subscriptions and the users 'getting' the publications from the destination queue.

Considerations for model queues:

To open a model queue, you must be able to open both the model queue itself and the dynamic queue to which it resolves. Define generic RACF profiles for dynamic queues, including dynamic queues used by IBM MQ utilities.

When you open a model queue, IBM MQ security makes two queue security checks:

1. Are you authorized to access the model queue?
2. Are you authorized to access the dynamic queue to which the model queue resolves?

If the dynamic queue name contains a trailing asterisk (*) character, this * is replaced by a character string generated by IBM MQ, to create a dynamic queue with a unique name. However, because the whole name, including this generated string, is used for checking authority, you should define generic profiles for these queues.

For example, an **MQOPEN** call uses a model queue name of CREDIT.CHECK.REPLY.MODEL and a dynamic queue name of CREDIT.REPLY.* on queue manager (or queue-sharing group) MQSP.

To do this, you must issue the following RACF commands to define the necessary queue profiles:

```
RDEFINE MQQUEUE MQSP.CREDIT.CHECK.REPLY.MODEL
RDEFINE MQQUEUE MQSP.CREDIT.REPLY.**
```

You must also issue the corresponding RACF PERMIT commands to allow the user access to these profiles.

A typical dynamic queue name created by an **MQOPEN** is something like CREDIT.REPLY.A346EF00367849A0. The precise value of the last qualifier is unpredictable; this is why you should use generic profiles for such queue names.

A number of IBM MQ utilities put messages on dynamic queues. You should define profiles for the following dynamic queue names, and provide RACF UPDATE access to the relevant user IDs (see “User IDs for security checking” on page 617 for the correct user IDs):

```
SYSTEM.CSQUTIL.*  (used by CSQUTIL)
SYSTEM.CSQOREXX.* (used by the operations and control panels)
SYSTEM.CSQXCMD.*  (used by the channel initiator when processing CSQINPX)
CSQ4SAMP.*        (used by the WebSphere MQ supplied samples)
```

You might also consider defining a profile to control use of the dynamic queue name used by default in the application programming copy members. The IBM MQ-supplied copybooks contain a default *DynamicQName*, which is CSQ.*. This enables an appropriate RACF profile to be established.

Note: Do not allow application programmers to specify a single * for the dynamic queue name. If you do, you must define an hlq.** profile in the MQQUEUE class, and you would have to give it wide-ranging access. This means that this profile could also be used for other non-dynamic queues that do not have a more specific RACF profile. Your users could, therefore, gain access to queues you do not want them to access.

Close options on permanent dynamic queues:

If an application opens a permanent dynamic queue that was created by another application and then attempts to delete that queue with an **MQCLOSE** option, some extra security checks are applied when the attempt is made.

Table 45. Access levels for close options on permanent dynamic queues

MQCLOSE option	RACF access level required to hlq.queuename
MQCO_DELETE	ALTER
MQCO_DELETE_PURGE	ALTER

Security and remote queues:

When a message is put on a remote queue, the queue security that is implemented by the local queue manager depends on how the remote queue is specified when it is opened.

The following rules are applied:

1. If the remote queue has been defined on the local queue manager through the IBM MQ DEFINE QREMOTE command, the queue that is checked is the name of the remote queue. For example, if a remote queue is defined on queue manager MQS1 as follows:
   ```
   DEFINE QREMOTE(BANK7.CREDIT.REFERENCE)
          RNAME(CREDIT.SCORING.REQUEST)
          RQMNAME(BNK7)
          XMITQ(BANK1.TO.BANK7)
   ```
   In this case, a profile for BANK7.CREDIT.REFERENCE must be defined in the MQQUEUE class.
2. If the *ObjectQMgrName* for the request does not resolve to the local queue manager, a security check is carried out against the resolved (remote) queue manager name except in the case of a cluster queue where the check is made against the cluster queue name.

   For example, the transmission queue BANK1.TO.BANK7 is defined on queue manager MQS1. An **MQPUT1** request is then issued on MQS1 specifying *ObjectName* as BANK1.INTERBANK.TRANSFERS and an *ObjectQMgrName* of BANK1.TO.BANK7. In this case, the user performing the request must have access to BANK1.TO.BANK7.
3. If you make an **MQPUT** request to a queue and specify `ObjectQMgrName` as the name of an alias of the local queue manager, only the queue name is checked for security, not that of the queue manager.

When the message gets to the remote queue manager it might be subject to additional security processing. For more information, see "Security for remote messaging" on page 463.

Dead-letter queue security:

Special considerations apply to the dead-letter queue, because many users must be able to put messages on it, but access to retrieve messages must be tightly restricted. You can achieve this by applying different RACF authorities to the dead-letter queue and an alias queue.

Undelivered messages can be put on a special queue called the dead-letter queue. If you have sensitive data that could possibly end up on this queue, you must consider the security implications of this because you do not want unauthorized users to retrieve this data.

Each of the following must be allowed to put messages onto the dead-letter queue:
- Application programs.
- The channel initiator address space and any MCA user IDs. (If the RESLEVEL profile is not present, or is defined so that channel user IDs are checked, the channel user ID also needs authority to put messages on the dead-letter queue.)
- CKTI, the IBM MQ-supplied CICS task initiator.
- CSQQTRMN, the IBM MQ-supplied IMS trigger monitor.

The only application that can retrieve messages from the dead-letter queue should be a 'special' application that processes these messages. However, a problem arises if you give applications RACF UPDATE authority to the dead-letter queue for **MQPUT** s because they can then automatically retrieve messages from the queue using **MQGET** calls. You cannot disable the dead-letter queue for get operations because, if you do, not even the 'special' applications could retrieve the messages.

One solution to this problem is set up a two-level access to the dead-letter queue. CKTI, message channel agent transactions or the channel initiator address space, and 'special' applications have direct access; other applications can only access the dead-letter queue through an alias queue. This alias is defined to allow applications to put messages on the dead-letter queue, but not to get messages from it.

This is how it might work:
1. Define the real dead-letter queue with attributes PUT(ENABLED) and GET(ENABLED), as shown in the sample thlqual.SCSQPROC(CSQ4INYG).
2. Give RACF UPDATE authority for the dead-letter queue to the following user IDs:
   - User IDs that the CKTI and the MCAs or channel initiator address space run under.
   - The user IDs associated with the 'special' dead-letter queue processing application.
3. Define an alias queue that resolves to the real dead-letter queue, but give the alias queue these attributes: PUT(ENABLED) and GET(DISABLED). Give the alias queue a name with the same stem as the dead-letter queue name but append the characters ".PUT" to this stem. For example, if the dead-letter queue name is hlq.DEAD.QUEUE, the alias queue name would be hlq.DEAD.QUEUE.PUT.
4. To put a message on the dead-letter queue, an application uses the alias queue. This is what your application must do:
   - Retrieve the name of the real dead-letter queue. To do this, it opens the queue manager object using **MQOPEN** and then issues an **MQINQ** to get the dead-letter queue name.
   - Build the name of the alias queue by appending the characters '.PUT' to this name, in this case, hlq.DEAD.QUEUE.PUT.
   - Open the alias queue, hlq.DEAD.QUEUE.PUT.
   - Put the message on the real dead-letter queue by issuing an **MQPUT** against the alias queue.
5. Give the user ID associated with the application RACF UPDATE authority to the alias, but no access (authority NONE) to the real dead-letter queue. This means that:
   - The application can put messages onto the dead-letter queue using the alias queue.
   - The application cannot get messages from the dead-letter queue using the alias queue because the alias queue is disabled for get operations.

The application cannot get any messages from the real dead-letter queue either because it does have the correct RACF authority.

Table 46 summarizes the RACF authority required for the various participants in this solution.

Table 46. RACF authority to the dead-letter queue and its alias

Associated user IDs	**Real dead-letter queue (hlq.DEAD.QUEUE)**	**Alias dead-letter queue (hlq.DEAD.QUEUE.PUT)**
MCA or channel initiator address space and CKTI	UPDATE	NONE
'Special' application (for dead-letter queue processing)	UPDATE	NONE
User-written application user IDs	NONE	UPDATE

If you use this method, the application cannot determine the maximum message length (MAXMSGL) of the dead-letter queue. This is because the MAXMSGL attribute cannot be retrieved from an alias queue. Therefore, your application should assume that the maximum message length is 100 MB, the maximum size IBM MQ for z/OS supports. The real dead-letter queue should also be defined with a MAXMSGL attribute of 100 MB.

Note: User-written application programs do not normally use alternate user authority to put messages on the dead-letter queue. This reduces the number of user IDs that have access to the dead-letter queue.

System queue security:

You must set up RACF access to allow certain user IDs access to particular system queues.

Many of the system queues are accessed by the ancillary parts of IBM MQ:
- The CSQUTIL utility
- The operations and control panels
- The channel initiator address space (including the Queued Pub/Sub Daemon)

The user IDs under which these run must be given RACF access to these queues, as shown in Table 47.

Table 47. Access required to the SYSTEM queues by IBM MQ

SYSTEM queue	**CSQUTIL**		**Operations and control panels**	**Channel initiator for distributed queuing**
SYSTEM.ADMIN.CHANNEL.EVENT	-	-	-	UPDATE
SYSTEM.BROKER.ADMIN.STREAM	-	-	-	ALTER
SYSTEM.BROKER.CONTROL.QUEUE	-	-	-	ALTER
SYSTEM.BROKER.DEFAULT.STREAM	-	-	-	ALTER
SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS	-	-	-	UPDATE
SYSTEM.CHANNEL.INITQ	-	-	-	UPDATE
SYSTEM.CHANNEL.SYNCQ	-	-	-	UPDATE
SYSTEM.CLUSTER.COMMAND.QUEUE	-	-	-	ALTER
SYSTEM.CLUSTER.REPOSITORY.QUEUE	-	-	-	UPDATE
SYSTEM.CLUSTER.TRANSMIT.QUEUE	-	-	-	ALTER
SYSTEM.COMMAND.INPUT	UPDATE	-	UPDATE	UPDATE
SYSTEM.COMMAND.REPLY.*	-	-	-	UPDATE
SYSTEM.COMMAND.REPLY.MODEL	UPDATE	-	UPDATE	UPDATE

Table 47. Access required to the SYSTEM queues by IBM MQ (continued)

SYSTEM queue	CSQUTIL	CSQ0UTIL	Operations and control panels	Channel initiator for distributed queuing
SYSTEM.CSQOREXX.*	-	-	UPDATE	-
SYSTEM.CSQUTIL.*	UPDATE	-	-	-
SYSTEM.CSQXCMD.*	-	-	-	UPDATE
SYSTEM.HIERARCHY.STATE	-	-	-	UPDATE
SYSTEM.INTER.QMGR.CONTROL	-	-	-	UPDATE
SYSTEM.INTER.QMGR.PUBS	-	-	-	UPDATE
SYSTEM.INTER.QMGR.FANREQ	-	-	-	UPDATE
SYSTEM.PROTECTION.ERROR.QUEUE	-	-	-	UPDATE
SYSTEM.PROTECTION.POLICY.QUEUE	-	UPDATE(1)	-	READ
SYSTEM.QSG.CHANNEL.SYNCQ	-	-	-	UPDATE
SYSTEM.QSG.TRANSMIT.QUEUE	-	-	-	UPDATE

API-resource security access quick reference:

A summary of the **MQOPEN**, **MQPUT1**, **MQSUB**, and **MQCLOSE** options and the access required by the different resource security types.

Table 48. MQOPEN, MQPUT1, MQSUB, and MQCLOSE options and the security authorization required. Callouts shown like this **(1)** refer to the notes following this table.

	Minimum RACF access level required			
RACF class:	**MXTOPIC**	**MQQUEUE or MXQUEUE (1)**	**MQADMIN or MXADMIN**	**MQADMIN or MXADMIN**
RACF profile:	**(15 or 16)**	**(2)**	**(3)**	**(4)**
MQOPEN option				
MQOO_INQUIRE		READ **(5)**	No check	No check
MQOO_BROWSE		READ	No check	No check
MQOO_INPUT_*		UPDATE	No check	No check
MQOO_SAVE_ALL_CONTEXT **(6)**		UPDATE	No check	No check
MQOO_OUTPUT (USAGE=NORMAL) **(7)**		UPDATE	No check	No check
MQOO_PASS_IDENTITY_CONTEXT **(8)**		UPDATE	READ	No check
MQOO_PASS_ALL_CONTEXT **(8) (9)**		UPDATE	READ	No check
MQOO_SET_IDENTITY_CONTEXT **(8) (9)**		UPDATE	UPDATE	No check
MQOO_SET_ALL_CONTEXT **(8) (10)**		UPDATE	CONTROL	No check
MQOO_OUTPUT (USAGE (XMITQ) **(11)**		UPDATE	CONTROL	No check
MQOO_OUTPUT (topic object)	UPDATE **(16)**			
MQOO_OUTPUT (alias queue to topic object)	UPDATE **(16)**	UPDATE		
MQOO_SET		ALTER	No check	No check
MQOO_ALTERNATE_USER_AUTHORITY		**(12)**	**(12)**	UPDATE
MQPUT1 option				
Put on a normal queue **(7)**		UPDATE	No check	No check

Table 48. MQOPEN, MQPUT1, MQSUB, and MQCLOSE options and the security authorization required (continued). Callouts shown like this **(1)** refer to the notes following this table.

	Minimum RACF access level required			
RACF class:	**MXTOPIC**	**MQQUEUE or MXQUEUE (1)**	**MQADMIN or MXADMIN**	**MQADMIN or MXADMIN**
RACF profile:	**(15 or 16)**	**(2)**	**(3)**	**(4)**
MQPMO_PASS_IDENTITY_CONTEXT		UPDATE	READ	No check
MQPMO_PASS_ALL_CONTEXT		UPDATE	READ	No check
MQPMO_SET_IDENTITY_CONTEXT		UPDATE	UPDATE	No check
MQPMO_SET_ALL_CONTEXT		UPDATE	CONTROL	No check
MQOO_OUTPUT Put on a transmission queue **(11)**		UPDATE	CONTROL	No check
MQOO_OUTPUT (topic object)	UPDATE **(16)**			
MQOO_OUTPUT (alias queue to topic object)	UPDATE **(16)**	UPDATE		
MQPMO_ALTERNATE_USER_AUTHORITY		**(13)**	**(13)**	UPDATE
MQCLOSE option				
MQCO_DELETE **(14)**		ALTER	No check	No check
MQCO_DELETE_PURGE **(14)**		ALTER	No check	No check
MQCO_REMOVE_SUB	ALTER **(15)**			
MQSUB option				
MQSO_CREATE	ALTER **(15)**	**(17)**	**(18)**	
MQSO_ALTER	ALTER **(15)**	**(17)**	**(18)**	
MQSO_RESUME	READ **(15)**	**(17)**	No check	
MQSO_ALTERNATE_USER_AUTHORITY				UPDATE
MQSO_SET_IDENTITY_CONTEXT			**(18)**	

Note:

1. This option is not restricted to queues. Use the MQNLIST or MXNLIST class for namelists, and the MQPROC or MXPROC class for processes.
2. Use RACF profile: hlq.resourcename
3. Use RACF profile: hlq.CONTEXT.queuename
4. Use RACF profile: hlq.ALTERNATE.USER. `alternateuserid`

   `alternateuserid` is the user identifier that is specified in the *AlternateUserId* field of the object descriptor. Note that up to 12 characters of the *AlternateUserId* field are used for this check, unlike other checks where only the first 8 characters of a user identifier are used.
5. No check is made when opening the queue manager for inquiries.
6. MQOO_INPUT_* must be specified as well. This is valid for a local, model or alias queue.
7. This check is done for a local or model queue that has a *Usage* queue attribute of MQUS_NORMAL, and also for an alias or remote queue (that is defined to the connected queue manager.) If the queue is a remote queue that is opened specifying an *ObjectQMgrName* (not the name of the connected queue manager) explicitly, the check is carried out against the queue with the same name as *ObjectQMgrName* (which must be a local queue with a *Usage* queue attribute of MQUS_TRANSMISSION).
8. MQOO_OUTPUT must be specified as well.

9. MQOO_PASS_IDENTITY_CONTEXT is implied as well by this option.
10. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT and MQOO_SET_IDENTITY_CONTEXT are implied as well by this option.
11. This check is done for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened.
12. At least one of MQOO_INQUIRE, MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT or MQOO_SET must be specified as well. The check carried out is the same as that for the other options specified.
13. The check carried out is the same as that for the other options specified.
14. This applies only for permanent dynamic queues that have been opened directly, that is, not opened through a model queue. No security is required to delete a temporary dynamic queue.
15. Use RACF profile hlq.SUBSCRIBE.topicname.
16. Use RACF profile hlq.PUBLISH.topicname.
17. If on the MQSUB request you specified a destination queue for the publications to be sent to, then a security check is carried out against that queue to ensure that you have put authority to that queue.
18. If on the MQSUB request, with MQSO_CREATE or MQSO_ALTER options specified, you want to set any of the identity context fields in the MQSD structure, you also need to specify the MQSO_SET_IDENTITY_CONTEXT option and you also need the appropriate authority to the context profile for the destination queue.

Profiles for topic security

If topic security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to those profiles.

The concept of topic security within a topic tree is described in Publish/subscribe security.

If topic security is active, you must perform the following actions:
- Define profiles in the **MXTOPIC** or **GMXTOPIC** classes.
- Permit the necessary groups or user IDs access to these profiles, so that they can issue IBM MQ API requests that use topics.

Profiles for topic security take the form:

```
hlq.SUBSCRIBE.topicname
hlq.PUBLISH.topicname
```

where
- `hlq` is either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name).
- `topicname` is the name of the topic administration node in the topic tree, associated either with the topic being subscribed to through an MQSUB call, or being published to through an MQOPEN call.

A profile prefixed by the queue manager name controls access to a single topic on that queue manager. A profile prefixed by the queue-sharing group name controls access to one or more topics with that topic name on all queue managers within the queue-sharing group. This access can be overridden on an individual queue manager by defining a queue-manager level profile for that topic on that queue manager.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

Subscribe

To subscribe to a topic, you need access to both the topic you are trying to subscribe to, and the target queue for the publications.

When you issue an MQSUB request, the following security checks take place:
- Whether you have the appropriate level of access to subscribe to that topic, and also that the target queue (if specified) is opened for output
- Whether you have the appropriate level of access to that target queue.

Table 49. Access level required for topic security to subscribe

Action	**Access level required**
MQSUB to a topic	RACF access required to `hlq.SUBSCRIBE.topicname` profile in MXTOPIC class
MQSO_CREATE and MQSO_ALTER	ALTER
MQSO_RESUME	READ
MQSUB - additional authority to non-managed destination queues.	RACF access required to `hlq.CONTEXT.queuename` profile in MQADMIN or MXADMIN class
MQSO_CREATE, MQSO_ALTER, and MQSO_RESUME	UPDATE
	RACF access required to `hlq.queuename` profile in MQQUEUE or MXQUEUE class
MQSO_CREATE and MQSO_ALTER	UPDATE
	RACF access required to `hlq.ALTERNATE.USER.alternateuserid` profile in MQADMIN or MXADMIN class
MQSO_ALTERNATE_USER_AUTHORITY	UPDATE

Considerations for managed queues for subscriptions

A security check is carried out to see if you are allowed to subscribe to the topic. However, no security checks are carried out when the managed queue is created, or to determine if you have access to put messages to this destination queue.

You cannot close delete a managed queue.

The model queues used are: `SYSTEM.DURABLE.MODEL.QUEUE` and `SYSTEM.NDURABLE.MODEL.QUEUE`.

The managed queues created from these model queues are of the form `SYSTEM.MANAGED.DURABLE.A346EF00367849A0` and `SYSTEM.MANAGED.NDURABLE.A346EF0036785EA0` where the last qualifier is unpredictable.

Do not give any user access to these queues. The queues can be protected using generic profiles of the form `SYSTEM.MANAGED.DURABLE.*` and `SYSTEM.MANAGED.NDURABLE.*` with no authorities granted.

Messages can be retrieved from these queues using the handle returned on the MQSUB request.

If you explicitly issue an MQCLOSE call for a subscription with the MQCO_REMOVE_SUB option specified, and you did not create the subscription you are closing under this handle, a security check is performed at the time of closure to ensure that you have the correct authority to perform the operation.

Table 50. Access level required to profiles for topic security for closure of a subscribe operation

Action	Access level required
MQCLOSE (of a subscription)	RACF access required to *hlq.SUBSCRIBE.topicname* profile in MXTOPIC class
MQCO_REMOVE_SUB	ALTER

Publish

To publish on a topic you need access to the topic and, if you are using alias queues, to the alias queue as well.

Table 51. Access level required to profiles for topic security for a publish operation

Action	Access level required
MQOPEN (of a topic)	RACF access required to *hlq.PUBLISH.topicname* profile in MXTOPIC class
MQOO_OUTPUT or MQPUT1	UPDATE
MQOPEN (Alias queue to topic)	RACF access required to *hlq.queuename* profile in MQQUEUE or MXQUEUE class for the alias queue
MQOO_OUTPUT or MQPUT1	UPDATE

For details of how topic security operates when an alias queue that resolves to a topic name is opened for publish, see "Considerations for alias queues that resolve to topics for a publish operation."

When you consider alias queues used for destination queues for PUT or GET restrictions, see "Considerations for alias queues" on page 584.

If the RACF access level that an application has to a topic security profile is changed, the changes take effect only for any new object handles obtained (that is, a new MQSUB or MQOPEN) for that topic. Those handles already in existence at the time of the change retain their existing access to the topic. Also, existing subscribers retain their access to any subscriptions that they have already made.

Considerations for alias queues that resolve to topics for a publish operation

When you issue an MQOPEN or MQPUT1 call for an alias queue that resolves to a topic, IBM MQ makes two resource checks:

- The first one against the alias queue name specified in the object descriptor (MQOD) on the MQOPEN or MQPUT1 call.
- The second against the topic to which the alias queue resolves

You must be aware that this behavior is different from the behavior you get when alias queues resolve to other queues. You need the correct access to both profiles in order for the publish action to proceed.

System topic security

Many of the system topics are accessed by the ancillary parts of IBM MQ, for example the channel initiator address space.

The user IDs under which this runs must be given RACF access to these queues, as shown in Table 52 on page 594.

Table 52. Access required to the SYSTEM topics

SYSTEM topic	Profile	Channel Initiator for Distributed queuing
SYSTEM.BROKER.ADMIN.STREAM	PUBLISH.topicname	UPDATE
SYSTEM.BROKER.ADMIN.STREAM	SUBSCRIBE.topicname	ALTER

Profiles for processes

If process security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to those profiles.

If process security is active, you must:

- Define profiles in the **MQPROC** or **GMQPROC** classes if using uppercase profiles.
- Define profiles in the **MXPROC** or **GMXPROC** classes if using mixed case profiles.
- Permit the necessary groups or user IDs access to these profiles, so that they can issue IBM MQ API requests that use processes.

Profiles for processes take the form:

```
hlq.processname
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name), and `processname` is the name of the process being opened.

A profile prefixed by the queue manager name controls access to a single process definition on that queue manager. A profile prefixed by the queue-sharing group name controls access to one or more process definitions with that name on all queue managers within the queue-sharing group. This access can be overridden on an individual queue manager by defining a queue-manager level profile for that process definition on that queue manager.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

The following table shows the access required for opening a process.

Table 53. Access levels for process security

MQOPEN option	RACF access level required to hlq.processname
MQOO_INQUIRE	READ

For example, on queue manager MQS9, the RACF group INQVPRC must be able to inquire (**MQINQ**) on all processes starting with the letter V. The RACF definitions for this would be:

```
RDEFINE MQPROC MQS9.V* UACC(NONE)
PERMIT MQS9.V* CLASS(MQPROC) ID(INQVPRC) ACCESS(READ)
```

Alternate user security might also be active, depending on the open options specified when a process definition object is opened.

Profiles for namelists

If namelist security is active, you define profiles in the appropriate classes and give the necessary groups or user IDs access to these profiles.

If namelist security is active, you must:

- Define profiles in the **MQNLIST** or **GMQNLIST** classes if using uppercase profiles.
- Define profiles in the **MXNLIST** or **GMXNLIST** classes if using mixed case profiles.
- Permit the necessary groups or user IDs access to these profiles.

Profiles for namelists take the form:

```
hlq.namelistname
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name), and `namelistname` is the name of the namelist being opened.

A profile prefixed by the queue manager name controls access to a single namelist on that queue manager. A profile prefixed by the queue-sharing group name controls access to access to one or more namelists with that name on all queue managers within the queue-sharing group. This access can be overridden on an individual queue manager by defining a queue-manager level profile for that namelist on that queue manager.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

The following table shows the access required for opening a namelist.

Table 54. Access levels for namelist security

MQOPEN option	RACF access level required to hlq.namelistname
MQOO_INQUIRE	READ

For example, on queue manager (or queue-sharing group) PQM3, the RACF group DEPT571 must be able to inquire (**MQINQ**) on these namelists:

- All namelists starting with "DEPT571".
- PRINTER/DESTINATIONS/DEPT571
- AGENCY/REQUEST/QUEUES
- WAREHOUSE.BROADCAST

The RACF definitions to do this are:

```
RDEFINE MQNLIST PQM3.DEPT571.** UACC(NONE)
PERMIT PQM3.DEPT571.** CLASS(MQNLIST) ID(DEPT571) ACCESS(READ)

RDEFINE GMQNLIST NLISTS.FOR.DEPT571 UACC(NONE)
        ADDMEM(PQM3.PRINTER/DESTINATIONS/DEPT571,
               PQM3.AGENCY/REQUEST/QUEUES,
               PQM3.WAREHOUSE.BROADCAST)
PERMIT NLISTS.FOR.DEPT571 CLASS(GMQNLIST) ID(DEPT571) ACCESS(READ)
```

Alternate user security might be active, depending on the options specified when a namelist object is opened.

System namelist security

Many of the system namelists are accessed by the ancillary parts of IBM MQ:
- The CSQUTIL utility
- The operations and control panels
- The channel initiator address space (including the Queued Publish/Subscribe Daemon)

The user IDs under which these run must be given RACF access to these namelists, as shown in Table 55.

Table 55. Access required to the SYSTEM namelists by IBM MQ

SYSTEM namelist	CSQUTIL	Operations and control panels	Channel initiator for distributed queuing
SYSTEM.QPUBSUB.QUEUE.NAMELIST	-	-	READ
SYSTEM.QPUBSUB.SUBPOINT.NAMELIST	-	-	READ

Profiles for alternate user security

If alternate user security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to those profiles.

For more information about *AlternateUserId*, see AlternateUserID (MQCHAR12).

If alternate user security is active, you must:
- Define profiles in the MQADMIN or GMQADMIN classes if you are using uppercase profiles.
- Define profiles in the MXADMIN or GMXADMIN classes if you are using mixed case profiles.

Permit the necessary groups or user IDs access to these profiles, so that they can use the ALTERNATE_USER_AUTHORITY options when the object is opened.

Profiles for alternate user security can be specified at subsystem level or at queue-sharing group level and take the following form:

```
hlq.ALTERNATE.USER.alternateuserid
```

Where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name), and `alternateuserid` is the value of the *AlternateUserId* field in the object descriptor.

A profile prefixed by the queue manager name controls use of an alternative user ID on that queue manager. A profile prefixed by the queue-sharing group name controls use of an alternative user ID on all queue managers within the queue-sharing group. This alternative user ID can be used on any queue manager within the queue-sharing group by a user that has the correct access. This access can be overridden on an individual queue manager by defining a queue-manager level profile for that alternative user ID on that queue manager.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

The following table shows the access when specifying an alternative user option.

Table 56. Access levels for alternate user security

MQOPEN, MQSUB, or MQPUT1 option	RACF access level required
MQOO_ALTERNATE_USER_AUTHORITY MQSO_ALTERNATE_USER_AUTHORITY MQPMO_ALTERNATE_USER_AUTHORITY	UPDATE

In addition to alternate user security checks, other security checks for queue, process, namelist, and context security can also be made. The alternative user ID, if provided, is only used for security checks on queue, process definition, or namelist resources. For alternate user and context security checks, the user ID requesting that the check is used. For details about how user IDs are handled, see “User IDs for security checking” on page 617. For a summary table showing the open options and the security checks required when queue, context and alternate user security are all active, see Table 48 on page 589.

An alternative user profile gives the requesting user ID access to resources associated with the user ID specified in the alternative user ID. For example, the payroll server running under user ID PAYSERV on queue manager QMPY processes requests from personnel user IDs, all of which start with PS. To cause the work performed by the payroll server to be carried out under the user ID of the requesting user, alternative user authority is used. The payroll server knows which user ID to specify as the alternative user ID because the requesting programs generate messages using the MQPMO_DEFAULT_CONTEXT put message option. See “User IDs for security checking” on page 617 for more details about from where alternative user IDs are obtained.

The following example RACF definitions enable the server program to specify alternative user IDs starting with the characters PS:

```
RDEFINE MQADMIN QMPY.ALTERNATE.USER.PS* UACC(NONE)
PERMIT QMPY.ALTERNATE.USER.PS* CLASS(MQADMIN) ID(PAYSERV) ACCESS(UPDATE)
```

Note:

1. The `AlternateUserId` fields in the object descriptor and subscription descriptor are 12 bytes long. All 12 bytes are used in the profile checks, but only the first 8 bytes are used as the user ID by IBM MQ. If this user ID truncation is not desirable, application programs making the request must translate any alternative user ID over 8 bytes into something more appropriate.
2. If you specify MQOO_ALTERNATE_USER_AUTHORITY, MQSO_ALTERNATE_USER_AUTHORITY, or MQPMO_ALTERNATE_USER_AUTHORITY and you do not specify an `AlternateUserId` field in the object descriptor, a user ID of blanks is used. For the purposes of the alternate user security check the user ID used for the `AlternateUserId` qualifier is -BLANK-. For example `RDEF MQADMIN hlq.ALTERNATE.USER.-BLANK-`.

   If the user is allowed to access this profile, all further checks are made with a user ID of blanks. For details of blank user IDs, see “Blank user IDs and UACC levels” on page 625.

The administration of alternative user IDs is easier if you have a naming convention for user IDs that enables you to use generic alternative user profiles. If they do not, you can use the RACF RACVARS feature. For details about using RACVARS, see the *z/OS SecureWay Security Server RACF Security Administrator's Guide*.

When a message is put to a queue that has been opened with alternative user authority and the context of the message has been generated by the queue manager, the MQMD_USER_IDENTIFIER field is set to the alternative user ID.

Profiles for context security

IBM MQ uses profiles for controlling access to the context information specific to a particular message. The context is contained within the message descriptor (MQMD).

Using profiles for context security

If context security is active, you must:

- Define a profile in the **MQADMIN** class if using uppercase profiles.
- Define profile in the **MXADMIN** class if using mixed case profiles.

The profile is called `hlq.CONTEXT.queuename`, where:

hlq Can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name).

queuename
Can be either the full name of the queue you want to define the context profile for, or a generic profile.

A profile prefixed by the queue manager name, and with ** specified as the queue name, allows control for context security on all queues belonging to that queue manager. This can be overridden on an individual queue by defining a queue level profile for context on that queue.

A profile prefixed by the queue-sharing group name, and with ** specified as the queue name, allows control for context on all queues belonging to the queue managers within the queue-sharing group. This can be overridden on an individual queue manager by defining a queue-manager level profile for context on that queue manager, by specifying a profile prefixed by the queue manager name. It can also be overridden on an individual queue by specifying a profile suffixed with the queue name.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

You must give the necessary groups or user IDs access to this profile. The following table shows the access level required, depending on the specification of the context options when the queue is opened.

Table 57. Access levels for context security

MQOPEN or MQPUT1 option	RACF access level required to hlq.CONTEXT.queuename
MQPMO_NO_CONTEXT	No context security check
MQPMO_DEFAULT_CONTEXT	No context security check
MQOO_SAVE_ALL_CONTEXT	No context security check
MQOO_PASS_IDENTITY_CONTEXT MQPMO_PASS_IDENTITY_CONTEXT	READ
MQOO_PASS_ALL_CONTEXT MQPMO_PASS_ALL_CONTEXT	READ
MQOO_SET_IDENTITY_CONTEXT MQPMO_SET_IDENTITY_CONTEXT	UPDATE

Table 57. Access levels for context security (continued)

MQOPEN or MQPUT1 option	**RACF access level required to hlq.CONTEXT.queuename**
MQOO_SET_ALL_CONTEXT MQPMO_SET_ALL_CONTEXT	CONTROL
MQOO_OUTPUT or MQPUT1 (USAGE(XMITQ))	CONTROL
MQSUB option	
MQSO_SET_IDENTITY_CONTEXT **(Note 2)**	UPDATE

Note:

1. The user IDs used for distributed queuing require CONTROL access to `hlq.CONTEXT.queuename` to put messages on the destination queue. See "User IDs used by the channel initiator" on page 620 for information about the user IDs used.
2. If on the MQSUB request, with MQSO_CREATE or MQSO_ALTER options specified, you want to set any of the identity context fields in the MQSD structure, you need to specify the MQSO_SET_IDENTITY_CONTEXT option. You require also, the appropriate authority to the context profile for the destination queue.

If you put commands on the system-command input queue, use the default context put message option to associate the correct user ID with the command.

For example, the IBM MQ-supplied utility program CSQUTIL can be used to offload and reload messages in queues. When offloaded messages are restored to a queue, the CSQUTIL utility uses the MQOO_SET_ALL_CONTEXT option to return the messages to their original state. In addition to the queue security required by this open option, context authority is also required. For example, if this authority is required by the group BACKGRP on queue manager MQS1, this would be defined by:

```
RDEFINE MQADMIN MQS1.CONTEXT.** UACC(NONE)
PERMIT MQS1.CONTEXT.** CLASS(MQADMIN) ID(BACKGRP) ACCESS(CONTROL)
```

Depending on the options specified, and the types of security performed, other types of security checks might also occur when the queue is opened. These include queue security (see "Profiles for queue security" on page 581), and alternate user security (see "Profiles for alternate user security" on page 596). For a summary table showing the open options and the security checks required when queue, context and alternate user security are all active, see Table 48 on page 589.

System queue context security

Many of the system queues are accessed by the ancillary parts of IBM MQ, for example the channel initiator address space.

The user IDs under which this runs must be given RACF access to these queues, as shown in Table 58 on page 600.

Table 58. Access required to the SYSTEM queues for context operations

SYSTEM queue	Channel Initiator for Distributed queuing
SYSTEM.BROKER.CONTROL.QUEUE	CONTROL
SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS	CONTROL
SYSTEM.CHANNEL.SYNCQ	CONTROL
SYSTEM.CLUSTER.TRANSMIT.QUEUE	CONTROL

Profiles for command security

To enable security checking for commands, add profiles to the MQCMDS class. The profile names are based on the MQSC commands but control both MQSC and PCF commands. Profiles can apply to a queue manager or a queue-sharing group.

If you want security checking for commands (so you have not defined the command security switch profile hlq.NO.CMD.CHECKS) you must add profiles to the MQCMDS class.

The same security profiles control both MQSC and PCF commands. The names of the RACF profiles for command security checking are based on the MQSC command names themselves. These profiles take the form:

```
hlq.verb.pkw
```

Where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name), `verb` is the verb part of the command name, for example ALTER, and `pkw` is the object type, for example QLOCAL for a local queue.

Thus, the profile name for the ALTER QLOCAL command in subsystem CSQ1 is:

```
CSQ1.ALTER.QLOCAL
```

You can use generic profiles to protect sets of commands so that you have fewer profiles to maintain and, therefore, fewer access lists. Consider creating a generic profile that applies to all commands not protected by a more specific profile. Define this profile with UACC(NONE) and grant ALTER access only to the RACF groups containing administrators. You might then create a generic profile applicable to all DISPLAY commands and grant widespread access to it. Between these extremes, you might identify groups of users needing access to certain sets of commands, in which case you can create profiles for those sets and grant access to RACF groups representing those classes of user. Avoid giving users access to commands they do not require: Apply the principle of least privilege, so that users only have access to the commands that are required for their jobs.

A profile prefixed by the queue manager name controls the use of the command on that queue manager. A profile prefixed by the queue-sharing group name controls the use of the command on all queue managers within the queue-sharing group. This access can be overridden on an individual queue manager by defining a queue-manager level profile for that command on that queue manager.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

By setting up command profiles at queue manager level, a user can be restricted from issuing commands on a particular queue manager. Alternatively, you can define one profile for a queue-sharing group for each command verb, and all security checks take place against that profile instead of individual queue managers.

If both subsystem security and queue-sharing group security are active and a local profile is not found, a command security check is performed to see if the user has access to a queue-sharing group profile.

If you use the CMDSCOPE attribute to route a command to other queue managers in a queue-sharing group, security is checked on each queue manager where the command is run, but not necessarily on the queue manager where the command is entered.

Table 59 shows, for each IBM MQ MQSC command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Table 60 on page 605 shows, for each IBM MQ PCF command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Table 59. MQSC commands, profiles, and their access levels

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
ALTER AUTHINFO	hlq.ALTER.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
ALTER BUFFPOOL	hlq.ALTER.BUFFPOOL	ALTER	No check	-
ALTER CFSTRUCT	hlq.ALTER.CFSTRUCT	ALTER	No check	-
ALTER CHANNEL	hlq.ALTER.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
ALTER NAMELIST	hlq.ALTER.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
ALTER PROCESS	hlq.ALTER.PROCESS	ALTER	hlq.PROCESS.process	ALTER
ALTER PSID	hlq.ALTER.PSID	ALTER	No check	-
ALTER QALIAS	hlq.ALTER.QALIAS	ALTER	hlq.QUEUE.queue	ALTER
ALTER QLOCAL	hlq.ALTER.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
ALTER QMGR	hlq.ALTER.QMGR	ALTER	No check	-
ALTER QMODEL	hlq.ALTER.QMODEL	ALTER	hlq.QUEUE.queue	ALTER
ALTER QREMOTE	hlq.ALTER.QREMOTE	ALTER	hlq.QUEUE.queue	ALTER
ALTER SECURITY	hlq.ALTER.SECURITY	ALTER	No check	-
ALTER SMDS	hlq.ALTER.SMDS	ALTER	No check	-
ALTER STGCLASS	hlq.ALTER.STGCLASS	ALTER	No check	-
ALTER SUB	hlq.ALTER.SUB	ALTER	No check	-
ALTER TOPIC	hlq.ALTER.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
ALTER TRACE	hlq.ALTER.TRACE	ALTER	No check	-
ARCHIVE LOG	hlq.ARCHIVE.LOG	CONTROL	No check	-
BACKUP CFSTRUCT	hlq.BACKUP.CFSTRUCT	CONTROL	No check	-
CLEAR QLOCAL	hlq.CLEAR.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
CLEAR TOPICSTR 3 on page 605	hlq.CLEAR.TOPICSTR	ALTER	hlq.TOPIC.topic	ALTER
DEFINE AUTHINFO	hlq.DEFINE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
DEFINE BUFFPOOL	hlq.DEFINE.BUFFPOOL	ALTER	No check	-
DEFINE CFSTRUCT	hlq.DEFINE.CFSTRUCT	ALTER	No check	-
DEFINE CHANNEL	hlq.DEFINE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
DEFINE LOG	hlq.DEFINE.LOG	ALTER	No check	-
DEFINE MAXSMSGS	hlq.DEFINE.MAXSMSGS	ALTER	No check	-
DEFINE NAMELIST	hlq.DEFINE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER

Table 59. MQSC commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
DEFINE PROCESS	hlq.DEFINE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
DEFINE PSID	hlq.DEFINE.PSID	ALTER	No check	-
DEFINE QALIAS	hlq.DEFINE.QALIAS	ALTER	hlq.QUEUE.queue	ALTER
DEFINE QLOCAL	hlq.DEFINE.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
DEFINE QMODEL	hlq.DEFINE.QMODEL	ALTER	hlq.QUEUE.queue	ALTER
DEFINE QREMOTE	hlq.DEFINE.QREMOTE	ALTER	hlq.QUEUE.queue	ALTER
DEFINE STGCLASS	hlq.DEFINE.STGCLASS	ALTER	No check	-
DEFINE SUB	hlq.DEFINE.SUB	ALTER	No check	-
DEFINE TOPIC	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
DELETE AUTHINFO	hlq.DELETE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
DELETE BUFFPOOL	hlq.DELETE.BUFFPOOL	ALTER	No check	-
DELETE CFSTRUCT	hlq.DELETE.CFSTRUCT	ALTER	No check	-
DELETE CHANNEL	hlq.DELETE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
DELETE NAMELIST	hlq.DELETE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
DELETE PROCESS	hlq.DELETE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
DELETE PSID	hlq.DELETE.PSID	ALTER	No check	-
DELETE QALIAS	hlq.DELETE.QALIAS	ALTER	hlq.QUEUE.queue	ALTER
DELETE QLOCAL	hlq.DELETE.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
DELETE QMODEL	hlq.DELETE.QMODEL	ALTER	hlq.QUEUE.queue	ALTER
DELETE QREMOTE	hlq.DELETE.QREMOTE	ALTER	hlq.QUEUE.queue	ALTER
DELETE STGCLASS	hlq.DELETE.STGCLASS	ALTER	No check	-
DELETE SUB	hlq.DELETE.SUB	ALTER	No check	-
DELETE TOPIC	hlq.DELETE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
DISPLAY ARCHIVE 1 on page 604	hlq.DISPLAY.ARCHIVE	READ	No check	-
DISPLAY AUTHINFO	hlq.DISPLAY.AUTHINFO	READ	No check	-
DISPLAY CFSTATUS	hlq.DISPLAY.CFSTATUS	READ	No check	-
DISPLAY CFSTRUCT	hlq.DISPLAY.CFSTRUCT	READ	No check	-
DISPLAY CHANNEL	hlq.DISPLAY.CHANNEL	READ	No check	-
DISPLAY CHINIT	hlq.DISPLAY.CHINIT	READ	No check	-
DISPLAY CHLAUTH	hlq.DISPLAY.CHLAUTH	READ	No check	-
DISPLAY CHSTATUS	hlq.DISPLAY.CHSTATUS	READ	No check	-
DISPLAY CLUSQMGR	hlq.DISPLAY.CLUSQMGR	READ	No check	-
DISPLAY CMDSERV	hlq.DISPLAY.CMDSERV	READ	No check	-
DISPLAY CONN 1 on page 604	hlq.DISPLAY.CONN	READ	No check	-
DISPLAY GROUP	hlq.DISPLAY.GROUP	READ	No check	-

Table 59. MQSC commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
DISPLAY LOG 1 on page 604	hlq.DISPLAY.LOG	READ	No check	-
DISPLAY MAXSMSGS	hlq.DISPLAY.MAXSMSGS	READ	No check	-
DISPLAY NAMELIST	hlq.DISPLAY.NAMELIST	READ	No check	-
DISPLAY PROCESS	hlq.DISPLAY.PROCESS	READ	No check	-
DISPLAY PUBSUB	hlq.DISPLAY.PUBSUB	READ	No check	-
DISPLAY QALIAS	hlq.DISPLAY.QALIAS	READ	No check	-
DISPLAY QCLUSTER	hlq.DISPLAY.QCLUSTER	READ	No check	-
DISPLAY QLOCAL	hlq.DISPLAY.QLOCAL	READ	No check	-
DISPLAY QMGR	hlq.DISPLAY.QMGR	READ	No check	-
DISPLAY QMODEL	hlq.DISPLAY.QMODEL	READ	No check	-
DISPLAY QREMOTE	hlq.DISPLAY.QREMOTE	READ	No check	-
DISPLAY QSTATUS	hlq.DISPLAY.QSTATUS	READ	No check	-
DISPLAY QUEUE	hlq.DISPLAY.QUEUE	READ	No check	-
DISPLAY SBSTATUS	hlq.DISPLAY.SBSTATUS	READ	No check	-
DISPLAY SMDS	hlq.DISPLAY.SMDS	READ	No check	-
DISPLAY SMDSCONN	hlq.DISPLAY.SMDSCONN	READ	No check	-
DISPLAY SUB	hlq.DISPLAY.SUB	READ	No check	-
DISPLAY SECURITY	hlq.DISPLAY.SECURITY	READ	No check	-
DISPLAY STGCLASS	hlq.DISPLAY.STGCLASS	READ	No check	-
DISPLAY SYSTEM 1 on page 604	hlq.DISPLAY.SYSTEM	READ	No check	-
DISPLAY THREAD	hlq.DISPLAY.THREAD	READ	No check	-
DISPLAY TPSTATUS	hlq.DISPLAY.TPSTATUS	READ	No check	-
DISPLAY TOPIC	hlq.DISPLAY.TOPIC	READ	No check	-
DISPLAY TPSTATUS	hlq.DISPLAY.TPSTATUS	READ	No check	-
DISPLAY TRACE	hlq.DISPLAY.TRACE	READ	No check	-
DISPLAY USAGE 1 on page 604	hlq.DISPLAY.USAGE	READ	No check	-
MOVE QLOCAL	hlq.MOVE.QLOCAL	ALTER	hlq.QUEUE.from-queue hlq.QUEUE.to-queue	ALTER
PING CHANNEL	hlq.PING.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
RECOVER BSDS	hlq.RECOVER.BSDS	CONTROL	No check	-
RECOVER CFSTRUCT	hlq.RECOVER.CFSTRUCT	CONTROL	No check	-
REFRESH CLUSTER	hlq.REFRESH.CLUSTER	ALTER	No check	-
REFRESH QMGR	hlq.REFRESH.QMGR	ALTER	No check	-
REFRESH SECURITY	hlq.REFRESH.SECURITY	ALTER	No check	-
RESET CFSTRUCT	hlq.RESET.CFSTRUCT	CONTROL	No check	-

Table 59. MQSC commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
RESET CHANNEL	hlq.RESET.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
RESET CLUSTER	hlq.RESET.CLUSTER	CONTROL	No check	-
RESET QMGR	hlq.RESET.QMGR	CONTROL	No check	-
RESET QSTATS	hlq.RESET.QSTATS	CONTROL	hlq.QUEUE.queue	CONTROL
RESET SMDS	hlq.RESET.SMDS	CONTROL	No check	-
RESET TPIPE	hlq.RESET.TPIPE	CONTROL	No check	-
RESOLVE CHANNEL	hlq.RESOLVE.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
RESOLVE INDOUBT	hlq.RESOLVE.INDOUBT	CONTROL	No check	-
RESUME QMGR	hlq.RESUME.QMGR	CONTROL	No check	-
RVERIFY SECURITY	hlq.RVERIFY.SECURITY	ALTER	No check	-
SET ARCHIVE	hlq.SET.ARCHIVE	CONTROL	No check	-
SET CHLAUTH	hlq.SET.CHLAUTH	CONTROL	No check	-
SET LOG	hlq.SET.LOG	CONTROL	No check	-
SET SYSTEM	hlq.SET.SYSTEM	CONTROL	No check	-
START CHANNEL	hlq.START.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
START CHINIT 4 on page 605	hlq.START.CHINIT	CONTROL	No check	-
START CMDSERV	hlq.START.CMDSERV	CONTROL	No check	-
START LISTENER	hlq.START.LISTENER	CONTROL	No check	-
START QMGR	None 2	-	-	-
START SMDSCONN	hlq.START.SMDSCONN	CONTROL	No check	-
START TRACE	hlq.START.TRACE	CONTROL	No check	-
STOP CHANNEL	hlq.STOP.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
STOP CHINIT	hlq.STOP.CHINIT	CONTROL	No check	-
STOP CMDSERV	hlq.STOP.CMDSERV	CONTROL	No check	-
STOP LISTENER	hlq.STOP.LISTENER	CONTROL	No check	-
STOP QMGR	hlq.STOP.QMGR	CONTROL	No check	-
STOP SMDSCONN	hlq.STOP.SMDSCONN	CONTROL	No check	-
STOP TRACE	hlq.STOP.TRACE	CONTROL	No check	-
SUSPEND QMGR	hlq.SUSPEND.QMGR	CONTROL	No check	-

Notes:

1. These commands might be issued internally by the queue manager; no authority is checked in these cases.
2. IBM MQ does not check the authority of the user who issues the START QMGR command. However, you can use RACF, or your alternative security facilities to control access to the START xxxxMSTR command that is issued as a result of the START QMGR command. This is done by controlling access to the MVS.START.STC.xxxxMSTR profile in the RACF operator commands (OPERCMDS) class. For

details of this procedure, see the *z/OS SecureWay Security Server RACF Security Administrator's Guide*. If you use this technique, and an unauthorized user tries to start the queue manager, it terminates with a reason code of 00F30216.

3. The **`hlq.TOPIC.topic`** resource refers to the Topic object derived from the TOPICSTR. For more details, see "Publish/subscribe security" on page 821
4. At releases prior to IBM MQ for z/OS V6, the security check was for MVS.START.STC.CSQ1CHIN. At IBM MQ for z/OS V6 and later, the resource name has an additional JOBNAME qualifier appended to it. This can cause problems when starting the channel initiator.

   To resolve the problem replace MVS.START.STC. *ssid* CHIN with a profile for a resource named MVS.START.STC. *ssid* CHIN **.*** or MVS.START.STC. *ssid* CHIN. *ssid* CHIN where *ssid* is the subsystem id for the queue manager. This requires RACF UPDATE authority. For more details, see the z/OS product documentation for *Operation planning, MVS Commands, RACF Access Authorities, and Resource Names*.

   The START for *ssid* MSTR does not include the JOBNAME= parameter. For consistency, you might want to update the profile for MVS.START.STC.ssidMSTR to MVS.START.STC.ssidMSTR.*.

Table 60. PCF commands, profiles, and their access levels

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Backup CF Structure	hlq.BACKUP.CFSTRUCT	CONTROL	No check	-
Change Authentication Information Object	hlq.ALTER.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Change CF Structure	hlq.ALTER.CFSTRUCT	ALTER	No check	-
Change Channel	hlq.ALTER.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Change Namelist	hlq.ALTER.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Change Process	hlq.ALTER.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Change Queue	hlq.ALTER.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Change Queue Manager	hlq.ALTER.QMGR	ALTER	No check	-
Change Security	hlq.ALTER.SECURITY	ALTER	No check	-
Change SMDS	hlq.ALTER.SMDS	ALTER	No check	-
Change Storage Class	hlq.ALTER.STGCLASS	ALTER	No check	-
Change Subscription	hlq.ALTER.SUB	ALTER	No check	-
Change Topic	hlq.ALTER.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Clear Queue	hlq.CLEAR.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
Clear Topic String 1 on page 608	hlq.CLEAR.TOPICSTR	ALTER	hlq.TOPIC.topic	ALTER
Copy Authentication Information Object	hlq.DEFINE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Copy CF Structure	hlq.DEFINE.CFSTRUCT	ALTER	No check	-
Copy Channel	hlq.DEFINE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Copy Namelist	hlq.DEFINE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Copy Process	hlq.DEFINE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Copy Queue	hlq.DEFINE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Copy Subscription	hlq.DEFINE.SUB	ALTER	No check	-

Table 60. PCF commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Copy Storage Class	hlq.DEFINE.STGCLASS	ALTER	No check	-
Copy Topic	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Create Authentication Information Object	hlq.DEFINE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Create CF Structure	hlq.DEFINE.CFSTRUCT	ALTER	No check	-
Create Channel	hlq.DEFINE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Create Namelist	hlq.DEFINE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Create Process	hlq.DEFINE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Create Queue	hlq.DEFINE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Create Storage Class	hlq.DEFINE.STGCLASS	ALTER	No check	-
Create Subscription	hlq.DEFINE.SUB	ALTER	No check	-
Create Topic	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Delete Authentication Information Object	hlq.DELETE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Delete CF Structure	hlq.DELETE.CFSTRUCT	ALTER	No check	-
Delete Channel	hlq.DELETE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Delete Namelist	hlq.DELETE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Delete Process	hlq.DELETE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Delete Queue	hlq.DELETE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Delete Storage Class	hlq.DELETE.STGCLASS	ALTER	No check	-
Delete Subscription	hlq.DELETE.SUB	ALTER	No check	-
Delete Topic	hlq.DELETE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Inquire Archive	hlq.DISPLAY.ARCHIVE	READ	No check	-
Inquire Authentication Information Object	hlq.DISPLAY.AUTHINFO	READ	No check	-
Inquire Authentication Information Object Names	hlq.DISPLAY.AUTHINFO	READ	No check	-
Inquire CF Structure	hlq.DISPLAY.CFSTRUCT	READ	No check	-
Inquire CF Structure Names	hlq.DISPLAY.CFSTRUCT	READ	No check	-
Inquire CF Structure Status	hlq.DISPLAY.CFSTATUS	READ	No check	-
Inquire Channel	hlq.DISPLAY.CHANNEL	READ	No check	-
Inquire Channel Authentication Records	hlq.DISPLAY.CHLAUTH	READ	No check	-
Inquire Channel Initiator	hlq.DISPLAY.CHINIT	READ	No check	-
Inquire Channel Names	hlq.DISPLAY.CHANNEL	READ	No check	-
Inquire Channel Status	hlq.DISPLAY.CHSTATUS	READ	No check	-

Table 60. PCF commands, profiles, and their access levels (continued)

Command	**Command profile for MQCMDS**	**Access level for MQCMDS**	**Command resource profile for MQADMIN or MXADMIN**	**Access level for MQADMIN or MXADMIN**
Inquire Cluster Queue Manager	hlq.DISPLAY.CLUSQMGR	READ	No check	-
Inquire Connection	hlq.DISPLAY.CONNPCF	READ	No check	-
Inquire Group	hlq.DISPLAY.GROUP	READ	No check	-
Inquire Log	hlq.DISPLAY.LOG	READ	No check	-
Inquire Namelist	hlq.DISPLAY.NAMELIST	READ	No check	-
Inquire Namelist Names	hlq.DISPLAY.NAMELIST	READ	No check	-
Inquire Process	hlq.DISPLAY.PROCESS	READ	No check	-
Inquire Process Names	hlq.DISPLAY.PROCESS	READ	No check	-
Inquire Pub/Sub Status	hlq.DISPLAY.PUBSUB	READ	No check	-
Inquire Queue	hlq.DISPLAY.QUEUE	READ	No check	-
Inquire Queue Manager	hlq.DISPLAY.QMGR	READ	No check	-
Inquire Queue Names	hlq.DISPLAY.QUEUE	READ	No check	-
Inquire Queue Status	hlq.DISPLAY.QSTATUS	READ	No check	-
Inquire Security	hlq.DISPLAY.SECURITY	READ	No check	-
Inquire SMDS	hlq.DISPLAY.SMDS	READ	No check	-
Inquire SMDSCONN	hlq.DISPLAY.SMDSCONN	READ	No check	-
Inquire Storage Class	hlq.DISPLAY.STGCLASS	READ	No check	-
Inquire Storage Class Names	hlq.DISPLAY.STGCLASS	READ	No check	-
Inquire Subscription	hlq.INQUIRE.SUB	READ	No check	-
Inquire Subscription Status	hlq.INQUIRE.SBSTATUS	READ	No check	-
Inquire System	hlq.DISPLAY.SYSTEM	READ	No check	-
Inquire Topic	hlq.DISPLAY.TOPIC	READ	No check	-
Inquire Topic Names	hlq.DISPLAY.TOPIC	READ	No check	-
Inquire Topic Status	hlq.DISPLAY.TPSTATUS	READ	No check	-
Inquire Usage	hlq.DISPLAY.USAGE	READ	No check	-
Move Queue	hlq.MOVE.QLOCAL	ALTER	hlq.QUEUE.from-queue hlq.QUEUE.to-queue	ALTER
Ping Channel	hlq.PING.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Recover CF Structure	hlq.RECOVER.CFSTRUCT	CONTROL	No check	-
Refresh Cluster	hlq.REFRESH.CLUSTER	ALTER	No check	-
Refresh Queue Manager	hlq.REFRESH.QMGR	ALTER	No check	-
Refresh Security	hlq.REFRESH.SECURITY	ALTER	No check	-
Reset CF Structure	hlq.RESET.CFSTRUCT	CONTROL	No check	-
Reset Channel	hlq.RESET.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL

Table 60. PCF commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Reset Cluster	hlq.RESET.CLUSTER	CONTROL	No check	-
Reset Queue Manager	hlq.RESET.QMGR	CONTROL	No check	-
Reset Queue Statistics	hlq.RESET.QSTATS	CONTROL	hlq.QUEUE.queue	CONTROL
Reset SMDS	hlq.RESET.SMDS	CONTROL	No check	-
Resolve Channel	hlq.RESOLVE.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Resume Queue Manager	hlq.RESUME.QMGR	CONTROL	No check	-
Resume Queue Manager Cluster	hlq.RESUME.QMGR	CONTROL	No check	-
Reverify Security	hlq.RVERIFY.SECURITY	ALTER	No check	-
Set Archive	hlq.SET.ARCHIVE	CONTROL	No check	-
Set Channel Authentication Record	hlq.SET.CHLAUTH	CONTROL	No check	-
Set Log	hlq.SET.LOG	CONTROL	No check	-
Set System	hlq.SET.SYSTEM	CONTROL	No check	-
Start Channel	hlq.START.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Start Channel Initiator	hlq.START.CHINIT	CONTROL	No check	-
Start Channel Listener	hlq.START.LISTENER	CONTROL	No check	-
Start SMDS Connection	hlq.START.SMDSCONN	CONTROL	No check	-
Stop Channel	hlq.STOP.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Stop Channel Initiator	hlq.STOP.CHINIT	CONTROL	No check	-
Stop Channel Listener	hlq.STOP.LISTENER	CONTROL	No check	-
Stop SMDS Connection	hlq.STOP.SMDSCONN	CONTROL	No check	-
Suspend Queue Manager	hlq.SUSPEND.QMGR	CONTROL	No check	-
Suspend Queue Manager Cluster	hlq.SUSPEND.QMGR	CONTROL	No check	-

Notes:

1. The **`hlq.TOPIC.topic`** resource refers to the Topic object derived from the TOPICSTR. For more details, see "Publish/subscribe security" on page 821

Profiles for command resource security

If you have not defined the command resource security switch profile, because you want security checking for resources associated with commands, you must add resource profiles for each resource to the appropriate class. The same security profiles control both MQSC and PCF commands.

If you have not defined the command resource security switch profile, `hlq.NO.CMD.RESC.CHECKS`, because you want security checking for resources associated with commands, you must:

- Add a resource profile in the **MQADMIN** class, if using uppercase profiles, for each resource.
- Add a resource profile in the **MXADMIN** class, if using mixed case profiles, for each resource.

The same security profiles control both MQSC and PCF commands.

Profiles for command resource security checking take the form:

```
hlq.type.resourcename
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue-sharing group name).

A profile prefixed by the queue manager name controls access to the resources associated with commands on that queue manager. A profile prefixed by the queue-sharing group name controls access to the resources associated with commands on all queue managers within the queue-sharing group. This access can be overridden on an individual queue manager by defining a queue-manager level profile for that command resource on that queue manager.

If your queue manager is a member of a queue-sharing group and you are using both queue manager and queue-sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue-sharing group name.

For example, the RACF profile name for command resource security checking against the model queue CREDIT.WORTHY in subsystem CSQ1 is:

```
CSQ1.QUEUE.CREDIT.WORTHY
```

Because the profiles for all types of command resource are held in the MQADMIN class, the "type" part of the profile name is needed in the profile to distinguish between resources of different types that have the same name. The "type" part of the profile name can be CHANNEL, QUEUE, TOPIC, PROCESS, or NAMELIST. For example, a user might be authorized to define hlq.QUEUE.PAYROLL.ONE, but not authorized to define hlq.PROCESS.PAYROLL.ONE

If the resource type is a queue, and the profile is a queue-sharing group level profile, it controls access to one or more local queues within the queue sharing group, or access to a single shared queue from any queue manager in the queue-sharing group.

z/OS MQSC commands, profiles, and their access levels shows, for each IBM MQ MQSC command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

z/OS PCF commands, profiles, and their access levels shows, for each IBM MQ PCF command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Command resource security checking for alias queues and remote queues:

Alias queue and remote queues both provide indirection to another queue. Additional points apply when you consider security checking for these queues.

Alias queues

When you define an alias queue, command resource security checks are only performed against the name of the alias queue, not against the name of the target queue to which the alias resolves.

Alias queues can resolve to both local and remote queues. If you do not want to permit users access to certain local or remote queues, you must do both of the following:
1. Do not allow the users access to these local and remote queues.
2. Restrict the users from being able to define aliases for these queues. That is, prevent them from being able to issue DEFINE QALIAS and ALTER QALIAS commands.

Remote queues

When you define a remote queue, command resource security checks are performed only against the name of the remote queue. No checks are performed against the names of the queues specified in the RNAME or XMITQ attributes in the remote queue object definition.

The RESLEVEL security profile

You can define a special profile in the MQADMIN or MXADMIN class to control the number of user IDs checked for API-resource security. This profile is called the RESLEVEL profile. How this profile affects API-resource security depends on how you access IBM MQ.

When an application tries to connect to IBM MQ, IBM MQ checks the access that the user ID associated with the connection has to a profile in the MQADMIN or MXADMIN class called:

```
hlq.RESLEVEL
```

Where `hlq` can be either `ssid` (subsystem ID) or `qsg` (queue-sharing group ID).

The user IDs associated with each connection type are:
- The user ID of the connecting task for batch connections
- The CICS address space user ID for CICS connections
- The IMS region address space user ID for IMS connections
- The channel initiator address space user ID for channel initiator connections

Attention: RESLEVEL is a very powerful option; it can cause the bypassing of all resource security checks for a particular connection.

If you do not have a RESLEVEL profile defined, you must be careful that no other profile in the MQADMIN class matches hlq.RESLEVEL. For example, if you have a profile in MQADMIN called hlq.** and no hlq.RESLEVEL profile, beware of the consequences of the hlq.** profile because it is used for the RESLEVEL check.

Define an hlq.RESLEVEL profile and set the UACC to NONE, rather than have no RESLEVEL profile at all. Have as few users or groups in the access list as possible. For details about how to audit RESLEVEL access, see "Auditing considerations on z/OS" on page 637.

If you are using queue manager level security only, IBM MQ performs RESLEVEL checks against the `qmgr-name.RESLEVEL` profile. If you are using queue-sharing group level security only, IBM MQ performs RESLEVEL checks against the `qsg-name.RESLEVEL` profile. If you are using a combination of both queue manager and queue-sharing group level security, IBM MQ first checks for the existence of a RESLEVEL profile at queue manager level. If it does not find one, it checks for a RESLEVEL profile at queue-sharing group level.

If it cannot find a RESLEVEL profile, IBM MQ enables checking of both the job and task (or alternate user) ID for a CICS or an IMS connection. For a batch connection, IBM MQ enables checking of the job (or alternate) user ID. For the channel initiator, IBM MQ enables checking of the channel user ID and the MCA (or alternate) user ID.

If there is a RESLEVEL profile, the level of checking depends on the environment and access level for the profile.

Remember that if your queue manager is a member of a queue-sharing group and you do not define this profile at queue-manager level, there might be one defined at queue-sharing group level that will affect the level of checking.To activate the checking of two user IDs, you define a RESLEVEL profile (prefixed with either the queue manager name of the queue-sharing group name) with a UACC(NONE) and ensure that the relevant users do not have access granted against this profile.

When you consider the access that the channel initiator's user ID has to RESLEVEL, remember that the connection established by the channel initiator is also the connection used by the channels. A setting that causes the bypassing of all resource security checks for the channel initiator's user ID effectively bypasses security checks for all channels. If the channel initiator's user ID access to RESLEVEL is something other than NONE, then only one user ID (for an access level of READ or UPDATE) or no user IDs (for an access level of CONTROL or ALTER) is checked for access. If you grant the channel initiator's user ID an access level other than NONE to RESLEVEL, be sure that you understand the effect of this setting on the security checks done for channels.

Using the RESLEVEL profile means that normal security audit records are not taken. For example, if you put UAUDIT on a user, the access to the hlq.RESLEVEL profile in MQADMIN is not audited.

If you use the RACF WARNING option on the hlq.RESLEVEL profile, no RACF warning messages are produced for profiles in the RESLEVEL class.

Security checking for report messages such as CODs are controlled by the RESLEVEL profile associated with the originating application. For example, if a batch job's userid has CONTROL or ALTER authority to a RESLEVEL profile, then all resource checking performed by the batch job are bypassed, including the security check of report messages.

If you change the RESLEVEL profile, users must disconnect and connect again before the change takes place. (This includes stopping and restarting the channel initiator if the access that the distributed queuing address space user ID has to the RESLEVEL profile is changed.)

To switch RESLEVEL auditing off, use the RESAUDIT system parameter.

RESLEVEL and batch connections

By default, when an IBM MQ resource is being accessed through batch and batch-type connections, the user must be authorized to access that resource for the particular operation. You can bypass the security check by setting up an appropriate RESLEVEL definition.

Whether the user is checked or not is based on the user ID used at connect time, the same user ID used for the connection check.

For example, you can set up RESLEVEL so that when a user you trust accesses certain resources through a batch connection, no API-resource security checks are done; but when a user you do not trust tries to access the same resources, security checks are carried out as normal. You should set up RESLEVEL checking to bypass API-resource security checks only when you sufficiently trust the user and the programs run by that user.

The following table shows the checks made for batch connections.

Table 61. Checks made at different RACF access levels for batch connections

RACF access level	Level of checking
NONE	Resource checks performed
READ	Resource checks performed
UPDATE	Resource checks performed
CONTROL	No check.
ALTER	No check.

RESLEVEL and system functions

The application of RESLEVEL to the operation and control panels, and to CSQUTIL.

The operation and control panels and the CSQUTIL utility are batch-type applications that make requests to the queue manager's command server, and so they are subject to the considerations described in “RESLEVEL and batch connections.” You can use RESLEVEL to bypass security checking for the SYSTEM.COMMAND.INPUT and SYSTEM.COMMAND.REPLY.MODEL queues that they use, but not for the dynamic queues SYSTEM.CSQXCMD.*, SYSTEM.CSQOREXX.*, and SYSTEM.CSQUTIL.*.

The command server is an integral part of the queue manager and so does not have connection or RESLEVEL checking associated with it. To maintain security, therefore, the command server must confirm that the user ID of the requesting application has authority to open the queue being used for replies. For the operations and control panels, this is SYSTEM.CSQOREXX.*. For CSQUTIL, it is SYSTEM.CSQUTIL.*. Users must be authorized to use these queues, as described in “System queue security” on page 588, in addition to any RESLEVEL authorization they are given.

For other applications using the command server, it is the queue they name as their reply-to queue. Such other applications might deceive the command server into placing messages on unauthorized queues by passing (in the message context) a more trusted user ID than its own to the command server. To prevent this, use a CONTEXT profile to protect the identity context of messages placed on SYSTEM.COMMAND.INPUT.

RESLEVEL and CICS connections

By default, when an API-resource security check is made on a CICS connection, two user IDs are checked. You can change which user IDs are checked by setting up a RESLEVEL profile.

The first user ID checked is that of the CICS address space. This is the user ID on the job card of the CICS job, or the user ID assigned to the CICS started task by the z/OS STARTED class or the started procedures table. (It is not the CICS DFLTUSER.)

The second user ID checked is the user ID associated with the CICS transaction.

If one of these user IDs does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED. Both the CICS address space user ID and the user ID of the person running the CICS transaction must have access to the resource at the correct level.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested. See Table 62 for more information.

The user IDs checked depend on the user ID used at connection time, that is, the CICS address space user ID. This control enables you to bypass API-resource security checking for IBM MQ requests coming from one system (for example, a test system, TESTCICS,) but to implement them for another (for example, a production system, PRODCICS).

Note: If you set up your CICS address space user ID with the “trusted” attribute in the STARTED class or the RACF started procedures table ICHRIN03, this overrides any user ID checks for the CICS address space established by the RESLEVEL profile for your queue manager (that is, the queue manager does not perform the security checks for the CICS address space). For more information, see the *CICS Transaction Server for z/OS V3.2 RACF Security Guide*.

The following table shows the checks made for CICS connections.

Table 62. Checks made at different RACF access levels for CICS connections

RACF access level	Level of checking
NONE	IBM MQ checks the CICS address space user ID and the transaction user ID.
READ	IBM MQ checks the CICS address space user ID only.
UPDATE	If the transaction is defined to CICS with RESSEC(YES), IBM MQ checks the CICS address space user ID and the transaction user ID.
UPDATE	If the transaction is defined to CICS with RESSEC(NO), IBM MQ checks the CICS address space user ID only.
CONTROL or ALTER	IBM MQ does not check any user IDs.

RESLEVEL and IMS connections

By default, when an API-resource security check is made for an IMS connection, two user IDs are checked. You can change which user IDs are checked by setting up a RESLEVEL profile.

By default, when an API-resource security check is made for an IMS connection, two user IDs are checked to see if access is allowed to the resource.

The first user ID checked is that of the address space of the IMS region. This is taken from either the USER field from the job card or the user ID assigned to the region from the z/OS STARTED class or the started procedures table (SPT).

The second user ID checked is associated with the work being done in the dependent region. It is determined according to the type of the dependent region as shown in How the second user ID is determined for the IMS(tm) connection.

If either the first or second IMS user ID does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED.

The setting of IBM MQ RESLEVEL profiles cannot alter the user ID under which IMS transactions are scheduled from the IBM-supplied MQ-IMS trigger monitor program CSQQTRMN. This user ID is the PSBNAME of that trigger monitor, which by default is CSQQTRMN.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested. The possible checks are:

- Check the IMS region address space user ID and the second user ID or alternate user ID.
- Check IMS region address space user ID only.
- Do not check any user IDs.

The following table shows the checks made for IMS connections.

Table 63. Checks made at different RACF access levels for IMS connections

RACF access level	Level of checking
NONE	Check the IMS address space user ID and the IMS second user ID or alternate user ID.
READ	Check the IMS address space user ID.
UPDATE	Check the IMS address space user ID.
CONTROL	No check.
ALTER	No check.

RESLEVEL and the channel initiator connection

By default, when an API-resource security check is made by the channel initiator, two user IDs are checked. You can change which user IDs are checked by setting up a RESLEVEL profile.

By default, when an API-resource security check is made by the channel initiator, two user IDs are checked to see if access is allowed to the resource.

The user IDs checked can be that specified by the MCAUSER channel attribute, that received from the network, that of the channel initiator address space, or the alternate user ID for the message descriptor. Which user IDs are checked depends on the communication protocol you are using and the setting of the PUTAUT channel attribute. See "User IDs used by the channel initiator" on page 620 for more information.

If one of these user IDs does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested, and how many are checked.

The following table shows the checks made for the channel initiator's connection, and for all channels since they use this connection.

Table 64. Checks made at different RACF access levels for channel initiator connections

RACF access level	Level of checking
NONE	Check two user IDs.
READ	Check one user ID.
UPDATE	Check one user ID.
CONTROL	No check.
ALTER	No check.
Note: See "User IDs used by the channel initiator" on page 620 for a definition of the user IDs checked	

RESLEVEL and intra-group queuing

By default, when an API-resource security check is made by the intra-group queuing agent, two user IDs are checked to see if access is allowed to the resource. You can change which user IDs are checked by setting up an RESLEVEL profile.

The user IDs checked can be the user ID determined by the IGQUSER attribute of the receiving queue manager, the user ID of the queue manager within the queue-sharing group that put the message on to the SYSTEM.QSG.TRANSMIT.QUEUE, or the alternate user ID specified in the `UserIdentifier` field of the message descriptor of the message. See "User IDs used by the intra-group queuing agent" on page 624 for more information.

Because the intra-group queuing agent is an internal queue manager task, it does not issue an explicit connect request and runs under the user ID of the queue manager. The intra-group queuing agent starts at queue manager initialization. During the initialization of the intra-group queuing agent, IBM MQ checks the access that the user ID associated with the queue manager has to a profile in the MQADMIN class called:

```
hlq.RESLEVEL
```

This check is always performed unless the hlq.NO.SUBSYS.SECURITY switch has been set.

If there is no RESLEVEL profile, IBM MQ enables checking for two user IDs. If there is a RESLEVEL profile, the level of checking depends on the access level granted to the user ID of the queue manager for the profile. Checks made at different RACF(r) access levels for the intra-group queuing agent shows the checks made for the intra-group queuing agent.

Table 65. Checks made at different RACF access levels for the intra-group queuing agent

RACF access level	Level of checking
NONE	Check two user IDs.
READ	Check one user ID.
UPDATE	Check one user ID.
CONTROL	No check.
ALTER	No check.
Note: See “User IDs used by the intra-group queuing agent” on page 624 for a definition of the user IDs checked	

If the permissions granted to the RESLEVEL profile for the queue manager's user ID are changed, the intra-group queuing agent must be stopped and restarted to pick up the new permissions. Because there is no way to independently stop and restart the intra-group queuing agent, the queue manager must be stopped and restarted to achieve this.

RESLEVEL and the user IDs checked

Example of setting a RESLEVEL profile and granting access to it.

User ID checking against profile name for batch connections through User IDs checked against profile name for LU 6.2 and TCP/IP server-connection channels show how RESLEVEL affects which user IDs are checked for different MQI requests.

For example, you have a queue manager called QM66 with the following requirements:

- User WS21B is to be exempt from resource security.
- CICS started task WXNCICS running under address space user ID CICSWXN is to perform full resource checking only for transactions defined with RESSEC(YES).

To define the appropriate RESLEVEL profile, issue the following RACF command:

```
RDEFINE MQADMIN QM66.RESLEVEL UACC(NONE)
```

Then give the users access to this profile, using the following commands:

```
PERMIT QM66.RESLEVEL CLASS(MQADMIN) ID(WS21B) ACCESS(CONTROL)
PERMIT QM66.RESLEVEL CLASS(MQADMIN) ID(CICSWXN) ACCESS(UPDATE)
```

If you make these changes while the user IDs are connected to queue manager QM66, the users must disconnect and connect again before the change takes place.

If subsystem security is not active when a user connects but, while this user is still connected, subsystem security becomes active, full resource security checking is applied to the user. The user must reconnect to get the correct RESLEVEL processing.

User IDs for security checking

IBM MQ initiates security checks based on user IDs associated with users, terminals, applications, and other resources. This collection of topics lists which user IDs are used for each type of security check.

User IDs for connection security

The user ID used for connection security depends on the type of connection.

Connection type	User ID contents
Batch connection	The user ID of the connecting task. For example: • The TSO user ID • The user ID assigned to a batch job by the USER JCL parameter • The user ID assigned to a started task by the STARTED class or the started procedures table
CICS connection	The CICS address space user ID.
IMS connection	The IMS region address space user ID.
Channel initiator connection	The channel initiator address space user ID.

User IDs for command security and command resource security

The user ID used for command security or command resource security depends on where the command is issued from.

Issued from...	User ID contents
CSQINP1, CSQINP2, or CSQINPT	No check is made.
System command input queue	The user ID found in the `UserIdentifier` of the message descriptor of the message that contains the command. If the message does not contain a `UserIdentifier`, a user ID of blanks is passed to the security manager.
Console	The user ID signed onto the console. If the console is not signed on, the default user ID set by the CMDUSER system parameter in CSQ6SYSP. To issue commands from a console, the console must have the z/OS SYS AUTHORITY attribute.
SDSF/TSO console	TSO or job user ID.
Operations and control panels	TSO user ID. If you are going to use the operations and control panels, you must have the appropriate authority to issue the commands corresponding to the actions that you choose. In addition, you must have READ access to all the hlq.DISPLAY. *object* profiles in the MQCMDS class because the panels use the various DISPLAY commands to gather the information that they present.
MGCRE	If MGCRE is used with UTOKEN, the user ID in the UTOKEN. If MGCRE is issued without the UTOKEN, the TSO or job user ID is used.
CSQ0UTIL	Job user ID.
CSQUTIL	Job user ID.
CSQINPX	User ID of the channel initiator address space.

User IDs for resource security (MQOPEN, MQSUB, and MQPUT1)

This information shows the contents of the user IDs for normal and alternate user IDs for each type of connection. The number of checks is defined by the RESLEVEL profile. The user ID checked is that used for **MQOPEN**, **MQSUB**, or **MQPUT1** calls.

Note: All user ID fields are checked exactly as they are received. No conversions take place, and, for example, three user ID fields containing “Bob”, “BOB”, and “bob” are not equivalent.

User IDs checked for batch connections:

The user ID checked for a batch connection depends on how the task is run and whether an alternate user ID has been specified.

Table 66. User ID checking against profile name for batch connections

Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
No	-	JOB	JOB
Yes	JOB	JOB	ALT

Key:

ALT Alternate user ID.

JOB

- The user ID of a TSO or USS sign-on.
- The user ID assigned to a batch job.
- The user ID assigned to a started task by the STARTED class or the started procedures table.
- The user ID associated with the executing Db2 stored procedure

A Batch job is performing an **MQPUT1** to a queue called Q1 with RESLEVEL set to READ and alternate user ID checking turned off.

Checks made at different RACF(r) access levels for batch connections and User ID checking against profile name for batch connections show that the job user ID is checked against profile hlq.Q1.

User IDs checked for CICS connections:

The user IDs checked for CICS connections depend on whether one or two checks are to be carried out, and whether an alternate user ID is specified.

Table 67. User ID checking against profile name for CICS-type user IDs

Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
No, 1 check	-	ADS	ADS
No, 2 checks	-	ADS+TXN	ADS+TXN
Yes, 1 check	ADS	ADS	ADS
Yes, 2 checks	ADS+TXN	ADS+TXN	ADS+ALT

Key:

ALT Alternate user ID

ADS The user ID associated with the CICS batch job or, if CICS is running as a started task, through the STARTED class or the started procedures table.

TXN The user ID associated with the CICS transaction. This is normally the user ID of the terminal user who started the transaction. It can be the CICS DFLTUSER, a PRESET security terminal, or a manually signed-on user.

Determine the user IDs checked for the following conditions:
- The RACF access level to the RESLEVEL profile, for a CICS address space user ID, is set to NONE.
- An **MQOPEN** call is made against a queue with MQOO_OUTPUT and MQOO_PASS_IDENTITY_CONTEXT.

First, see how many CICS user IDs are checked based on the CICS address space user ID access to the RESLEVEL profile. From Table 62 on page 613 in topic "RESLEVEL and CICS connections" on page 613, two user IDs are checked if the RESLEVEL profile is set to NONE. Then, from Table 67 on page 618 on, these checks are carried out:
- The hlq.ALTERNATE.USER.userid profile is not checked.
- The hlq.CONTEXT.queuename profile is checked with both the CICS address space user ID and the CICS transaction user ID.
- The hlq.resourcename profile is checked with both the CICS address space user ID and the CICS transaction user ID.

This means that four security checks are made for this **MQOPEN** call.

User IDs checked for IMS connections:

The user IDs checked for IMS connections depend on whether one or two checks are to be performed, and whether an alternate user ID is specified. If a second user ID is checked, it depends on the type of dependent region and on which user IDs are available.

Table 68. User ID checking against profile name for IMS-type user IDs

Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
No, 1 check	-	REG	REG
No, 2 checks	-	REG+SEC	REG+SEC
Yes, 1 check	REG	REG	REG
Yes, 2 checks	REG+SEC	REG+SEC	REG+ALT

Key:

ALT Alternate user ID.

REG The user ID is normally set through the STARTED class or the started procedures table or, if IMS is running, from a submitted job, by the USER JCL parameter.

SEC The second user ID is associated with the work being done in a dependent region. It is determined according to Table 69 on page 620.

Table 69. How the second user ID is determined for the IMS connection

Types of dependent region	Hierarchy for determining the second user ID
• BMP message driven and successful GET UNIQUE issued. • IFP and GET UNIQUE issued. • MPP.	User ID associated with the IMS transaction if the user is signed on. LTERM name if available. PSBNAME.
• BMP message driven and successful GET UNIQUE not issued. • BMP not message driven. • IFP and GET UNIQUE not issued.	User ID associated with the IMS dependent region address space if this is not all blanks or all zeros. PSBNAME.

User IDs used by the channel initiator:

This collection of topics describes the user IDs used and checked for receiving channels and for client MQI requests issued over server-connection channels. Information is provided for TCP/IP and for LU6.2

You can use the PUTAUT parameter of the receiving channel definition to determine the type of security checking used. To get consistent security checking throughout your IBM MQ network, you can use the ONLYMCA and ALTMCA options.

You can use the DISPLAY CHSTATUS command to determine the user identifier used by the MCA.

Receiving channels using TCP/IP:

The user IDs checked depend on the PUTAUT option of the channel and on whether one or two checks are to be performed.

Table 70. User IDs checked against profile name for TCP/IP channels

PUTAUT option specified on receiver or requester channel	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
DEF, 1 check	-	CHL	CHL
DEF, 2 checks	-	CHL + MCA	CHL + MCA
CTX, 1 check	CHL	CHL	CHL
CTX, 2 checks	CHL + MCA	CHL + MCA	CHL + ALT
ONLYMCA, 1 check	-	MCA	MCA
ONLYMCA, 2 checks	-	MCA	MCA
ALTMCA, 1 check	MCA	MCA	MCA
ALTMCA, 2 checks	MCA	MCA	MCA + ALT

Key:

MCA (MCA user ID)
The user ID specified for the MCAUSER channel attribute at the receiver; if blank, the channel initiator address space user ID of the receiver or requester side is used.

CHL (Channel user ID)
On TCP/IP, security is not supported by the communication system for the channel. If the Secure Sockets Layer (SSL) is being used and a digital certificate has been flowed from the partner, the user ID associated with this certificate (if installed), or the user ID associated with a matching

filter found by using RACF Certificate Name Filtering (CNF), is used. If no associated user ID is found, or if SSL is not being used, the user ID of the channel initiator address space of the receiver or requester end is used as the channel user ID on channels defined with the PUTAUT parameter set to DEF or CTX.

Note: The use of RACF Certificate Name Filtering (CNF) allows you to assign the same RACF user ID to multiple remote users, for example all the users in the same organization unit, who would naturally all have the same security authority. This means that the server does not have to have a copy of the certificate of every possible remote end user across the world and greatly simplifies certificate management and distribution.

If the PUTAUT parameter is set to ONLYMCA or ALTMCA for the channel, the channel user ID is ignored and the MCA user ID of the receiver or requester is used. This also applies to TCP/IP channels using SSL.

ALT (Alternate user ID)
The user ID from the context information (that is, the `UserIdentifier` field) within the message descriptor of the message. This user ID is moved into the `AlternateUserID` field in the object descriptor before an **MQOPEN** or **MQPUT1** call is issued for the target destination queue.

Receiving channels using LU 6.2:

The user IDs checked depend on the PUTAUT option of the channel and on whether one or two checks are to be performed.

Table 71. User IDs checked against profile name for LU 6.2 channels

PUTAUT option specified on receiver or requester channel	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
DEF, 1 check	-	CHL	CHL
DEF, 2 checks	-	CHL + MCA	CHL + MCA
CTX, 1 check	CHL	CHL	CHL
CTX, 2 checks	CHL + MCA	CHL + MCA	CHL + ALT
ONLYMCA, 1 check	-	MCA	MCA
ONLYMCA, 2 checks	-	MCA	MCA
ALTMCA, 1 check	MCA	MCA	MCA
ALTMCA, 2 checks	MCA	MCA	MCA + ALT

Key:

MCA (MCA user ID)
The user ID specified for the MCAUSER channel attribute at the receiver; if blank, the channel initiator address space user ID of the receiver or requester side is used.

CHL (Channel user ID)

Requester-server channels
If the channel is started from the requester, there is no opportunity to receive a network user ID (the channel user ID).

If the PUTAUT parameter is set to DEF or CTX on the requester channel, the channel user ID is that of the channel initiator address space of the requester because no user ID is received from the network.

If the PUTAUT parameter is set to ONLYMCA or ALTMCA, the channel user ID is ignored and the MCA user ID of the requester is used.

Other channel types

If the PUTAUT parameter is set to DEF or CTX on the receiver or requester channel, the channel user ID is the user ID received from the communications system when the channel is initiated.

- If the sending channel is on z/OS, the channel user ID received is the channel initiator address space user ID of the sender.
- If the sending channel is on a different platform (for example, AIX or HP-UX), the channel user ID received is typically provided by the USERID parameter of the channel definition.

If the user ID received is blank, or no user ID is received, a channel user ID of blanks is used.

ALT (Alternate user ID)

The user ID from the context information (that is, the `UserIdentifier` field) within the message descriptor of the message. This user ID is moved into the `AlternateUserID` field in the object descriptor before an **MQOPEN** or **MQPUT1** call is issued for the target destination queue.

Client MQI requests:

Various user IDs can be used, depending on which user IDs and environment variables have been set. These user IDs are checked against various profiles, depending on the PUTAUT option used and whether an alternate user ID is specified.

This section describes the user IDs checked for client MQI requests issued over server-connection channels for TCP/IP and LU 6.2. The MCA user ID and channel user ID are as for the TCP/IP and LU 6.2 channels described in the previous sections.

For server-connection channels, the user ID received from the client is used if the MCAUSER attribute is blank.

See "Access control for clients" on page 467 for more information.

For client **`MQOPEN`**, **`MQSUB`**, and **`MQPUT1`** requests, use the following rules to determine the profile that is checked:

- If the request specifies alternate-user authority, a check is made against the *hlq*.ALTERNATE.USER. *userid* profile.
- If the request specifies context authority, a check is made against the *hlq*.CONTEXT. *queuename* profile.
- For all **`MQOPEN`**, **`MQSUB`**, and **`MQPUT1`** requests, a check is made against the *hlq.resourcename* profile.

When you have determined which profiles are checked, use the following table to determine which user IDs are checked against these profiles.

Table 72. User IDs checked against profile name for LU 6.2 and TCP/IP server-connection channels

PUTAUT option specified on server-connection channel	Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
DEF, 1 check	No	-	CHL	CHL
DEF, 1 check	Yes	CHL	CHL	CHL
DEF, 2 checks	No	-	CHL + MCA	CHL + MCA

Table 72. User IDs checked against profile name for LU 6.2 and TCP/IP server-connection channels (continued)

PUTAUT option specified on server-connection channel	Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
DEF, 2 checks	Yes	CHL + MCA	CHL + MCA	CHL + ALT
ONLYMCA, 1 check	No	-	MCA	MCA
ONLYMCA, 1 check	Yes	MCA	MCA	MCA
ONLYMCA, 2 checks	No	-	MCA	MCA
ONLYMCA, 2 checks	Yes	MCA	MCA	MCA + ALT

Key:

MCA (MCA user ID)
The user ID specified for the MCAUSER channel attribute at the server-connection; if blank, the channel initiator address space user ID is used.

CHL (Channel user ID)
On TCP/IP, security is not supported by the communication system for the channel. If the Secure Sockets Layer (SSL) is being used and a digital certificate has been flowed from the partner, the user ID associated with this certificate (if installed), or the user ID associated with a matching filter found by using RACF Certificate Name Filtering (CNF), is used. If no associated user ID is found, or if SSL is not being used, the user ID of the channel initiator address space is used as the channel user ID on channels defined with the PUTAUT parameter set to DEF or CTX.

Note: The use of RACF Certificate Name Filtering (CNF) allows you to assign the same RACF user ID to multiple remote users, for example all the users in the same organization unit, who would naturally all have the same security authority. This means that the server does not have to have a copy of the certificate of every possible remote end user across the world and greatly simplifies certificate management and distribution.

If the PUTAUT parameter is set to ONLYMCA or ALTMCA for the channel, the channel user ID is ignored and the MCA user ID of the server-connection channel is used. This also applies to TCP/IP channels using SSL.

ALT (Alternate user ID)
The user ID from the context information (that is, the `UserIdentifier` field) within the message descriptor of the message. This user ID is moved into the `AlternateUserID` field in the object or subscription descriptor before an **MQOPEN**, **MQSUB** or **MQPUT1** call is issued on behalf of the client application.

Channel initiator example:

An example of how user IDs are checked against RACF profiles.

A user performs an **MQPUT1** operation to a queue on queue manager QM01 that resolves to a queue called QB on queue manager QM02. The message is sent on a TCP/IP channel called QM01.TO.QM02. RESLEVEL is set to NONE, and the open is performed with alternate user ID and context checking. The receiver channel definition has PUTAUT(CTX) and the MCA user ID is set. Which user IDs are used on the receiving channel to put the message to queue QB?

Answer: Table 64 on page 615 shows that two user IDs are checked because RESLEVEL is set to NONE.

Table 70 on page 620 shows that, with PUTAUT set to CTX and 2 checks, the following user IDs are checked:
- The channel initiator user ID and the MCAUSER user ID are checked against the hlq.ALTERNATE.USER.userid profile.
- The channel initiator user ID and the MCAUSER user ID are checked against the hlq.CONTEXT.queuename profile.
- The channel initiator user ID and the alternate user ID specified in the message descriptor (MQMD) are checked against the hlq.Q2 profile.

User IDs used by the intra-group queuing agent:

The user IDs that are checked when the intra-group queuing agent opens destination queues are determined by the values of the IGQAUT and IGQUSER queue manager attributes.

The possible user IDs are:

Intra-group queuing user ID (IGQ)

The user ID determined by the IGQUSER attribute of the receiving queue manager. If this is set to blanks, the user ID of the receiving queue manager is used. However, because the receiving queue manager has authority to access all queues defined to it, security checks are not performed for the receiving queue manager's user ID. In this case:
- If only one user ID is to be checked and the user ID is that of the receiving queue manager, no security checks take place. This can occur when IGAUT is set to ONLYIGQ or ALTIGQ.
- If two user IDs are to be checked and one of the user IDs is that of the receiving queue manager, security checks take place for the other user ID only. This can occur when IGAUT is set to DEF, CTX, or ALTIGQ.
- If two user IDs are to be checked and both user IDs are that of the receiving queue manager, no security checks take place. This can occur when IGAUT is set to ONLYIGQ.

Sending queue manager user ID (SND)

The user ID of the queue manager within the queue-sharing group that put the message on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Alternate user ID (ALT)

The user ID specified in the *UserIdentifier* field in the message descriptor of the message.

Table 73. User IDs checked against profile name for intra-group queuing

IGQAUT option specified on receiving queue manager	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
DEF, 1 check	-	SND	SND
DEF, 2 checks	-	SND +IGQ	SND +IGQ
CTX, 1 check	SND	SND	SND
CTX, 2 checks	SND + IGQ	SND +IGQ	SND + ALT
ONLYIGQ, 1 check	-	IGQ	IGQ
ONLYIGQ, 2 checks	-	IGQ	IGQ
ALTIGQ, 1 check	-	IGQ	IGQ
ALTIGQ, 2 checks	IGQ	IGQ	IGQ + ALT

Key:

ALT	Alternate user ID.
IGQ	IGQ user ID.
SND	Sending queue manager user ID.

Blank user IDs and UACC levels

If a blank user ID occurs, a RACF undefined user is signed on. Do not grant wide-ranging access to the undefined user.

Blank user IDs can exist when a user is manipulating messages using context or alternate-user security, or when IBM MQ is passed a blank user ID. For example, a blank user ID is used when a message is written to the system-command input queue without context.

Note: A user ID of " * " (that is, an asterisk character followed by seven spaces) is treated as an undefined user ID.

IBM MQ passes the blank user ID to RACF and a RACF undefined user is signed on. All security checks then use the universal access (UACC) for the relevant profile. Depending on how you have set your access levels, the UACC might give the undefined user a wide-ranging access.

For example, if you issue this RACF command from TSO:

```
RDEFINE MQQUEUE Q.AVAILABLE.TO.EVERYONE UACC(UPDATE)
```

you define a profile that enables both z/OS-defined user IDs (that have not been put in the access list) and the RACF undefined user ID to put messages on, and get messages from, that queue.

To protect against blank user IDs you must plan your access levels carefully, and limit the number of people who can use context and alternate-user security. You must prevent people using the RACF undefined user ID from getting access to resources that they must not access. However, at the same time, you must allow access to people with defined user IDs. To do this, you can specify a user ID of asterisk (*) in a RACF command PERMIT, giving access to resources for all defined user IDs. Therefore all undefined user IDs (such as " * ") are denied access. For example, these RACF commands prevent the RACF undefined user ID from gaining access to the queue to put or get messages:

```
RDEFINE MQQUEUE Q.AVAILABLE.TO.RACF.DEFINED.USERS.ONLY UACC(NONE)
PERMIT Q.AVAILABLE.TO.RACF.DEFINED.USERS.ONLY CLASS(MQQUEUE) ACCESS(UPDATE) ID(*)
```

z/OS user IDs and Multi-Factor Authentication (MFA)

z/OS

IBM Multi-Factor Authentication for z/OS allows z/OS security administrators to enhance SAF authentication, by requiring identified users to use multiple authentication factors (for example, both a password and a cryptographic token) to sign on to a z/OS system. IBM MFA also provides support for time-based one time password generation technologies such as RSA SecureId.

For the most part, IBM MQ is unaware of how users have "logged on" to the CICS or batch systems that are driving IBM MQ work, the signed on user ID credential is associated with the z/OS task or address space and IBM MQ uses this for checking authorization to resources. User IDs enabled for MFA can be used for authorization to IBM MQ resources and authentication through pass tickets used with the CICS and IMS bridges.

Important: Special considerations apply however, when using applications, such as the MQ Explorer, which pass a user ID and password credentials on an MQCONNX API call with the *MQCSP_AUTH_USER_ID_AND_PWD* option. IBM MQ has no facility to pass an additional credential on this API request.

Limitations and potential workarounds are described in the following text.

MQ Explorer

The MQ Explorer cannot be used to log on to a z/OS system with a userid for which MFA is enabled because there is no facility for passing a second authentication factor from the MQ Explorer to z/OS.

Additionally, there are two different mechanisms used by the MQ Explorer to re-use a user ID and password credential, that need special attention when one time use passwords are in effect:

1. MQ Explorer has the capability to store passwords in an obfuscated format on the local machine for login at a later time. This capability must be disabled by having explorer prompt for a password each time a connection is made to the z/OS queue manager.

   To do this, use the following procedure:

   a. Select **Queue Managers**.
   b. From the list displayed, choose the queue manager you require and right click that queue manager.
   c. Select **Connection Details** from the menu list that appears.
   d. Select **Properties** from the next menu list and choose the **Userid** tab.

      Ensure that you select the **prompt for password** radio button.
2. Various operations in the MQ Explorer, such as browsing messages on queues, testing subscriptions, and so on, start a new thread which authenticates to IBM MQ using the credential first used at logon. Since the password credential cannot be re-used, you cannot use these operations.

There are two possible workarounds at the MFA configuration level for these issues:

- Use the application ID exclusion of MFA to exclude the IBM MQ tasks from MFA processing altogether.

  To do this, issue the following commands:

  1. `RDEFINE MFADEF MFABYPASS.USERID.<chinuser>`

     where `<chinuser>` is the channel initiator address space level user Id (associated with the channel initiator through the STC class)
  2. `PERMIT MFABYPASS.USERID.<chinuser> ACCESS(READ) ID(<explorer user>)`

  For more information on this approach, see Bypassing IBM MFA for applications.

- Use Out-of-band support on MFA, which was introduced with IBM MFA 1.2. With this approach, you pre-authenticate to the IBM MFA web server, and in addition to your user ID and password, specify additional authentication as determined through the policy. IBM MFA server generates a cache token credential that you then specify on the MQ Explorer authentication dialogue. The security administrator can allow this credential to be replayed for a reasonable period of time, so enabling normal MQ Explorer use.

For more information on this approach see Introduction to IBM MFA.

IBM MQ for z/OS security management

IBM MQ uses an in-storage table to hold information relating to each user and the access requests made by each user. To manage this table efficiently and to reduce the number of requests made from IBM MQ to the external security manager (ESM), a number of controls are available.

These controls are available through both the operations and control panels and IBM MQ commands.

User ID reverification

If the RACF definition of a user who is using IBM MQ resources has been changed, for example by connecting the user to a new group, you can tell the queue manager to sign this user on again the next time it tries to access an IBM MQ resource. You can do this by using the IBM MQ command RVERIFY SECURITY.

- User HX0804 is getting and putting messages to the PAYROLL queues on queue manager PRD1. However HX0804 now requires access to some of the PENSION queues on the same queue manager (PRD1).
- The data security administrator connects user HX0804 to the RACF group that allows access to the PENSION queues.
- So that HX0804 can access the PENSION queues immediately - that is, without shutting down queue manager PRD1, or waiting for HX0804 to time out - you must use the IBM MQ command:

```
RVERIFY SECURITY(HX0804)
```

Note: If you turn off user ID timeout for long periods of time (days or even weeks) while the queue manager is running, you must remember to run the RVERIFY SECURITY command for any users that have been revoked or deleted in that time.

User ID timeouts

You can make IBM MQ sign a user off a queue manager after a period of inactivity.

When a user accesses an IBM MQ resource, the queue manager tries to sign this user on to the queue manager (if subsystem security is active). This means that the user is authenticated to the ESM. This user remains signed on to IBM MQ until either the queue manager is shut down, or until the user ID is *timed out* (the authentication lapses) or reverified (reauthenticated).

When a user is timed out, the user ID is *signed off* within the queue manager and any security-related information retained for this user is discarded. The signing on and off of the user within the queue manager is not apparent to the application program or to the user.

Users are eligible for timeout when they have not used any IBM MQ resources for a predetermined amount of time. This time period is set by the MQSC ALTER SECURITY command.

Two values can be specified in the ALTER SECURITY command:

TIMEOUT
The time period in minutes that an unused user ID and its associated resources can remain within the IBM MQ queue manager.

INTERVAL
The time period in minutes between checks for user IDs and their associated resources, to determine whether the *TIMEOUT* has expired.

For example, if the *TIMEOUT* value is 30 and the *INTERVAL* value is 10, every 10 minutes IBM MQ checks user IDs and their associated resources to determine whether any have not been used for 30 minutes. If a timed-out user ID is found, that user ID is signed off within the queue manager. If any timed-out resource information associated with non-timed-out user IDs is found, that resource information is discarded. If you do not want to time out user IDs, set the *INTERVAL* value to zero. However, if the *INTERVAL* value is zero, storage occupied by user IDs and their associated resources is not freed until you issue a **REFRESH SECURITY** or **RVERIFY SECURITY** command.

Tuning this value can be important if you have many one-off users. If you set small interval and timeout values, resources that are no longer required are freed.

Note: If you use values for *INTERVAL* or *TIMEOUT* other than the defaults, you must reenter the command at every queue manager startup. You can do this automatically by putting the **ALTER SECURITY** command in the CSQINP1 data set for that queue manager.

Refreshing queue manager security on z/OS

IBM MQ for z/OS caches RACF data to improve performance. When you change certain security classes, you must refresh this cached information. Refresh security infrequently, for performance reasons. You can also choose to refresh only SSL security information.

When a queue is opened for the first time (or for the first time since a security refresh) IBM MQ performs a RACF check to obtain the user's access rights and places this information in the cache. The cached data includes user IDs and resources on which security checking has been performed. If the queue is opened again by the same user, the presence of the cached data means that IBM MQ does not have to issue RACF checks, which improves performance. The action of a security refresh is to discard any cached security information and so force IBM MQ to make a new check against RACF. Whenever you add, change or delete a RACF resource profile that is held in the MQADMIN, MXADMIN, MQPROC, MXPROC, MQQUEUE, MXQUEUE, MQNLIST, MXNLIST, or MXTOPIC class, you must tell the queue managers that use this class to refresh the security information that they hold. To do this, issue the following commands:

- The RACF SETROPTS RACLIST(classname) REFRESH command to refresh at the RACF level.
- The IBM MQ REFRESH SECURITY command to refresh the security information held by the queue manager. This command needs to be issued by each queue manager that accesses the profiles that have changed. If you have a queue-sharing group, you can use the command scope attribute to direct the command to all the queue managers in the group.

If you are using generic profiles in any of the IBM MQ classes, you must also issue normal RACF refresh commands if you change, add, or delete any generic profiles. For example, SETROPTS GENERIC(classname) REFRESH.

However, if a RACF resource profile is added, changed or deleted, and the resource to which it applies has not yet been accessed (so no information is cached), IBM MQ uses the new RACF information without a REFRESH SECURITY command being issued.

If RACF auditing is turned on, (for example, by using the RACF RALTER AUDIT(access-attempt (audit_access_level)) command), no caching takes place, and therefore IBM MQ refers directly to the RACF dataspace for every check. Changes are therefore picked up immediately and REFRESH SECURITY is not necessary to access the changes. You can confirm whether RACF auditing is on by using the RACF RLIST command. For example, you could issue the command

```
RLIST MQQUEUE (qmgr.SYSTEM.COMMAND.INPUT) GEN
```

and receive the results

```
CLASS      NAME
-----      ----
MQQUEUE    QP*.SYSTEM.COMMAND.*.** (G)
    AUDITING
    --------
    FAILURES(READ)
```

This indicates that auditing is set on. For more information, refer to the *z/OS Security Server RACF Auditor's Guide* and the *z/OS Security Server RACF Command Language Reference*.

Figure 70 on page 630 summarizes the situations in which security information is cached and in which cached information is used.

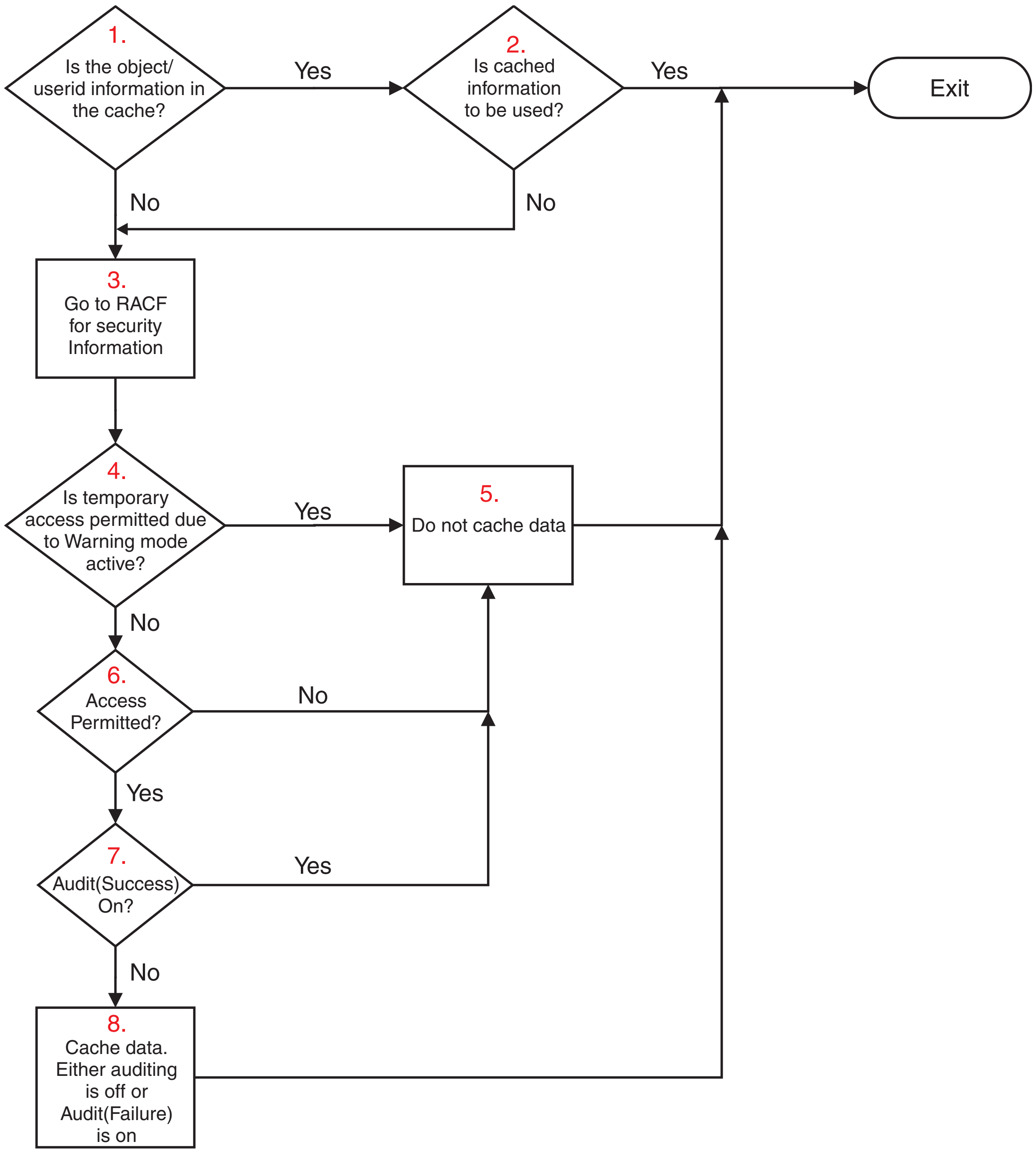

1.
Is the object/ userid information in the cache?
Yes
2.
Is cached information to be used?
Yes
Exit
No
No
3.
Go to RACF for security Information
4.
Is temporary access permitted due to Warning mode active?
Yes
5.
Do not cache data
No
6.
Access Permitted?
No
Yes
7.
Audit(Success) On?
Yes
No
8.
Cache data. Either auditing is off or Audit(Failure) is on

Figure 70. Logic flow for IBM MQ security caching

If you change your security settings by adding or deleting switch profiles in the MQADMIN or MXADMIN classes, use one of these commands to pick up these changes dynamically:

REFRESH SECURITY(*)

REFRESH SECURITY(MQADMIN)

REFRESH SECURITY(MXADMIN)

This means you can activate new security types, or deactivate them without having to restart the queue manager.

For performance reasons, these are the only classes affected by the REFRESH SECURITY command. You do *not* need to use REFRESH SECURITY if you change a profile in either the MQCONN or MQCMDS classes.

Note: A refresh of the MQADMIN or MXADMIN class is not required if you change a RESLEVEL security profile.

For performance reasons, use REFRESH SECURITY as infrequently as possible, ideally at off-peak times. You can minimize the number of security refreshes by connecting users to RACF groups that are already in the access list for IBM MQ profiles, rather than putting individual users in the access lists. In this way, you change the user rather than the resource profile. You can also RVERIFY SECURITY the appropriate user instead of refreshing security.

As an example of REFRESH SECURITY, suppose you define the new profiles to protect access to queues starting with INSURANCE.LIFE on queue manager PRMQ. You use these RACF commands:

```
RDEFINE MQQUEUE PRMQ.INSURANCE.LIFE.** UACC(NONE)
PERMIT PRMQ.INSURANCE.LIFE.** ID(LIFEGRP) ACCESS(UPDATE)
```

You must issue the following command to tell RACF to refresh the security information that it holds, for example:

```
SETROPTS RACLIST(MQQUEUE) REFRESH
```

Because these profiles are generic, you must tell RACF to refresh the generic profiles for MQQUEUE. For example:

```
SETROPTS GENERIC(MQQUEUE) REFRESH
```

Then you must use this command to tell queue manager PRMQ that the queue profiles have changed:

```
REFRESH SECURITY(MQQUEUE)
```

Refreshing SSL security

To refresh the cached view of the SSL Key Repository, issue the REFRESH SECURITY command with the option TYPE(SSL). This enables you to update some of your SSL settings without having to restart your channel initiator.

Displaying security status

To display the status of the security switches, and other security controls, issue the MQSC DISPLAY SECURITY command.

The following figure shows typical output of the DISPLAY SECURITY ALL command.

```
CSQH015I +CSQ1 Security timeout = 54 MINUTES
CSQH016I +CSQ1 Security interval = 12 MINUTES
CSQH030I +CSQ1 Security switches ...
CSQH034I +CSQ1 SUBSYSTEM: ON, 'SQ05.NO.SUBSYS.SECURITY' not found
CSQH032I +CSQ1 QMGR: ON, 'CSQ1.YES.QMGR.CHECKS' found
CSQH031I +CSQ1 QSG: OFF, 'SQ05.NO.QSG.CHECKS' found
CSQH031I +CSQ1 CONNECTION: OFF, 'CSQ1.NO.CONNECT.CHECKS' found
CSQH034I +CSQ1 COMMAND: ON, 'CSQ1.NO.COMMAND.CHECKS' not found
CSQH031I +CSQ1 CONTEXT: OFF, 'CSQ1.NO.CONTEXT.CHECKS' found
CSQH034I +CSQ1 ALTERNATE USER: ON, 'CSQ1.NO.ALTERNATE.USER.CHECKS' not found
CSQH034I +CSQ1 PROCESS: ON, 'CSQ1.NO.PROCESS.CHECKS' not found
CSQH034I +CSQ1 NAMELIST: ON, 'CSQ1.NO.NLIST.CHECKS' not found
CSQH034I +CSQ1 QUEUE: ON, 'CSQ1.NO.QUEUE.CHECKS' not found
CSQH034I +CSQ1 TOPIC: ON, 'CSQ1.NO.TOPIC.CHECKS' not found
CSQH031I +CSQ1 COMMAND RESOURCES: OFF, 'CSQ1.NO.CMD.RESC.CHECKS' found
CSQ9022I +CSQ1 CSQHPDTC ' DISPLAY SECURITY' NORMAL COMPLETION
```

Figure 71. Typical output from the DISPLAY SECURITY command

The example shows that the queue manager that replied to the command has subsystem, command, alternate user, process, namelist, and queue security active at queue manager level but not at queue-sharing group level. Connection, command resource, and context security are not active. It also shows that user ID timeouts are active, and that every 12 minutes the queue manager checks for user IDs that have not been used in this queue manager for 54 minutes and removes them.

Note: This command shows the current security status. It does not necessarily reflect the current status of the switch profiles defined to RACF, or the status of the RACF classes. For example, the switch profiles might have been changed since the last restart of this queue manager or REFRESH SECURITY command.

Security installation tasks for z/OS

After installing and customizing IBM MQ, authorize started task procedures to RACF, authorize access to various resources, and set up RACF definitions. Optionally, configure your system for SSL.

When IBM MQ is first installed and customized, you must perform these security-related tasks:

1. Set up IBM MQ data set and system security by:
   - Authorizing the queue manager started-task procedure xxxxMSTR and the distributed queuing started-task procedure xxxxCHIN to run under RACF.
   - Authorizing access to queue manager data sets.
   - Authorizing access to resources for those user IDs that will use the queue manager and utility programs.
   - Authorizing access for those queue managers that will use the coupling facility list structures.
   - Authorizing access for those queue managers that will use Db2.
2. Set up RACF definitions for IBM MQ security.
3. If you want to use the Secure Sockets Layer (SSL), prepare your system to use certificates and keys.

Setting up IBM MQ for z/OS data set security

There are many types of IBM MQ user. Use RACF to control their access to system data sets.

The possible users of IBM MQ data sets include the following entities:

- The queue manager itself.
- The channel initiator
- IBM MQ administrators, who need to create IBM MQ data sets, run utility programs, and similar tasks.
- Application programmers who need to use the IBM MQ-supplied copybooks, include data sets, macros, and similar resources.
- Applications involving one or more of:
  - Batch jobs
  - TSO users
  - CICS regions
  - IMS regions
- Data sets CSQOUTX and CSQSNAP
- Dynamic queues SYSTEM.CSQXCMD.*

For all these potential users, protect the IBM MQ data sets with RACF.

You must also control access to all your 'CSQINP' data sets.

RACF authorization of started-task procedures:

Some IBM MQ data sets are for the exclusive use of the queue manager. If you protect your IBM MQ data sets using RACF, you must also authorize the queue manager started-task procedure xxxxMSTR, and the distributed queuing started-task procedure xxxxCHIN, using RACF. To do this, use the STARTED class. Alternatively, you can use the started procedures table (ICHRIN03), but then you must perform an IPL of your z/OS system before the changes take effect.

For more information, see the *z/OS Security Server RACF System Programmer's Guide*.

The RACF user ID identified must have the required access to the data sets in the started-task procedure. For example, if you associate a queue manager started task procedure called CSQ1MSTR with the RACF user ID QMGRCSQ1, the user ID QMGRCSQ1 must have access to the z/OS resources accessed by the CSQ1 queue manager.

Also, the content of the GROUP field in the user ID of the queue manager must be the same as the content of the GROUP field in the STARTED profile for that queue manager. If the content in each GROUP field does not match then the appropriate user ID is prevented from entering the system. This situation causes IBM MQ to run with an undefined user ID and consequently close due to a security violation.

The RACF user IDs associated with the queue manager and channel initiator started task procedures must not have the TRUSTED attribute set.

Authorizing access to data sets:

The IBM MQ data sets should be protected so that no unauthorized user can run a queue manager instance, or gain access to any queue manager data. To do this, use normal z/OS RACF data set protection.

Table 74 summarizes the RACF access that the queue manager started task procedure must have to the different data sets.

Table 74. RACF access to data sets associated with a queue manager

RACF access	Data sets
READ	• thlqual.SCSQAUTH and thlqual.SCSQANLx (where x is the language letter for your national language). • The data sets referred to by CSQINP1, CSQINP2 and CSQXLIB in the queue manager's started task procedure.
UPDATE	• All page sets and log and BSDS data sets.
ALTER	• All archive log data sets.

Table 75 on page 634 summarizes the RACF access that the started task procedure for distributed queuing must have to the different data sets.

Table 75. RACF access to data sets associated with distributed queuing

RACF access	Data sets
READ	• thlqual.SCSQAUTH, thlqual.SCSQANLx (where x is the language letter for your national language), and thlqual.SCSQMVR1. • LE library data sets. • The data sets referred to by CSQXLIB and CSQINPX in the distributed queuing started task procedure.
UPDATE	• Data sets CSQOUTX and CSQSNAP

For more information, see the *z/OS Security Server RACF Security Administrator's Guide*.

Setting up IBM MQ for z/OS resource security

There are many types of IBM MQ user. Use RACF to control their access to IBM MQ resources.

The possible users of IBM MQ resources, such as queues and channels include the following entities:

- The queue manager itself.
- The channel initiator
- IBM MQ administrators, who need to create IBM MQ data sets, run utility programs, and similar tasks
- Application programmers who need to use the IBM MQ-supplied copybooks, include data sets, macros, and similar resources.
- Applications involving one or more of:
  - Batch jobs
  - TSO users
  - CICS regions
  - IMS regions
- Data sets CSQOUTX and CSQSNAP
- Dynamic queues SYSTEM.CSQXCMD.*

For all these potential users, protect the IBM MQ resources with RACF. In particular, note that the channel initiator needs access to various resources, as described in "Security considerations for the channel initiator on z/OS" on page 641, and so the user ID under which it runs must be authorized to access these resources.

If you are using a queue-sharing group, the queue manager might issue various commands internally, so the user ID it uses must be authorized to issue such commands. The commands are:

- DEFINE, ALTER, and DELETE for every object that has QSGDISP(GROUP)
- START and STOP CHANNEL for every channel used with CHLDISP(SHARED)

Configuring your z/OS system to use the Secure Sockets Layer (SSL)

z/OS

Use this topic as example of how to configure IBM MQ for z/OS with SSL using RACF commands.

If you want to use SSL for channel security, there are a number of tasks you need to perform on your system. (For details on using RACF commands for certificates and key repositories (key rings), see Working with SSL or TLS on z/OS .)

1. Create a key ring in RACF to hold all the keys and certificates for your system, using the RACF RACDCERT command. For example:

```
RACDCERT ID(CHINUSER) ADDRING(QM1RING)
```

   The ID must be either the channel initiator address space user ID or the user ID you want to own the key ring if it is to be a shared key ring.

2. Create a digital certificate for each queue manager, using the RACF RACDCERT command.

   The label of the certificate must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager or queue-sharing group appended. See Digital certificate labels for details. In this example it is `ibmWebSphereMQQM1`.

   For example:

```
RACDCERT ID(USERID) GENCERT
SUBJECTSDN(CN('username') O('IBM') OU('departmentname') C('England'))
WITHLABEL('ibmWebSphereMQQM1')
```

3. Connect the certificate in RACF to the key ring, using the RACF RACDCERT command. For example:

```
RACDCERT CONNECT(ID(USERID) LABEL('ibmWebSphereMQQM1') RING(QM1RING))
CONNECT ID(CHINUSER)
```

   You also need to connect any relevant signer certificates (from a certificate authority) to the key ring. That is, all certificate authorities for the SSL certificate of this queue manager and all certificate authorities for all SSL certificates that this queue manager communicates with. For example:

```
RACDCERT ID(CHINUSER)
CONNECT(CERTAUTH LABEL('My CA') RING(QM1RING) USAGE(CERTAUTH))
```

4. On each of your queue managers, use the IBM MQ ALTER QMGR command to specify the key repository that the queue manager needs to point to. For example, if the key ring is owned by the channel initiator address space:

```
ALTER QMGR SSLKEYR(QM1RING)
```

or if you are using a shared key ring:

```
ALTER QMGR SSLKEYR(userid/QM1RING)
```

where *userid* is the user ID that owns the shared key ring.

5. Certificate Revocation Lists (CRLs) allow the certificate authorities to revoke certificates that can no longer be trusted. CRLs are stored in LDAP servers. To access this list on the LDAP server, you first need to create an AUTHINFO object of AUTHTYPE CRLLDAP, using the IBM MQ DEFINE AUTHINFO command. For example:

```
DEFINE AUTHINFO(LDAP1)
AUTHTYPE(CRLLDAP)
CONNAME(ldap.server(389))
LDAPUSER('')
LDAPPWD('')
```

   In this example, the certificate revocation list is stored in a public area of the LDAP server, so the LDAPUSER and LDAPPWD fields are not necessary.

   Next, put your AUTHINFO object into a namelist, using the IBM MQ DEFINE NAMELIST command. For example:

```
DEFINE NAMELIST(LDAPNL) NAMES(LDAP1)
```

   Finally, associate the namelist with each queue manager, using the IBM MQ ALTER QMGR command. For example:

```
ALTER QMGR SSLCRLNL(LDAPNL)
```

6. Set up your queue manager to run SSL calls, using the IBM MQ ALTER QMGR command. This defines server subtasks that handle SSL calls only, which leaves the normal dispatchers to continue processing as normal without being affected by any SSL calls. You must have at least two of these subtasks. For example:

```
ALTER QMGR SSLTASKS(8)
```

   This change only takes effect when the channel initiator is restarted.

7. Specify the cipher specification to be used for each channel, using the IBM MQ DEFINE CHANNEL or ALTER CHANNEL command. For example:

```
ALTER CHANNEL(LDAPCHL)
CHLTYPE(SDR)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA)
```

Both ends of the channel must specify the same cipher specification.

Managing channel authentication records in a queue-sharing group

Channel authentication records apply to the queue manager that they are created on, they are not shared throughout the queue-sharing group. Therefore if all the queue managers in the queue sharing group are required to have the same rules, some management needs to be carried out to keep all the rules the consistent.

1. Always add the `CMDSCOPE(*)` option to all `SET CHLAUTH` commands. This will send the command to all running queue managers in the queue-sharing group
2. Use the `DISPLAY CHLAUTH` command with the `CMDSCOPE(*)` option and then analyze the responses to see if the records are the same from all queue managers. When an inconsistency is found a `SET CHLAUTH` command can be issued containing the same rule with `CMDSCOPE(*)` or `CMDSCOPE(`*`qmgr-name`*`)`.
3. Add a member to the queue manager's CSQINP2 concatenation (see Initialization commands for details) that has the full set of rules. These will be read as part of the queue manager's initialization process. If the `SET CHLAUTH` command uses `ACTION(ADD)` the rule will only be added if it didn't exist. Using `ACTION(REPLACE)` will replace an existing rule if it already exists or add it if it does not. The same member could then be placed in the CSQINP2 concatenation of all queue managers in the queue-sharing group.
4. Use the CSQUTIL utility (see Issuing commands to IBM MQ (COMMAND) for details) to extract the rules from one queue manager using either the `MAKEDEF` or `MAKEREP` option. Then replay the output using CSQUTIL into the target queue manager.

Related concepts:

Channel authentication records
To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records.

Auditing considerations on z/OS

The normal RACF auditing controls are available for conducting a security audit of a queue manager. IBM MQ does not gather any security statistics of its own. The only statistics are those that can be created by auditing.

RACF auditing can be based upon:

- User IDs
- Resource classes
- Profiles

For more details, see the *z/OS Security Server RACF Auditor's Guide*.

Note: Auditing degrades performance; the more auditing you implement, the more performance is degraded. This is also a consideration for the use of the RACF WARNING option.

Auditing RESLEVEL

Use the RESAUDIT system parameter to control the production of RESLEVEL audit records. RACF GENERAL audit records are produced.

Produce RESLEVEL audit records by setting the RESAUDIT system parameter to YES. If the RESAUDIT parameter is set to NO, audit records are not produced. For more details about setting this parameter, see Using CSQ6SYSP.

If RESAUDIT is set to YES, no normal RACF audit records are taken when the RESLEVEL check is made to see what access an address space user ID has to the hlq.RESLEVEL profile. Instead, IBM MQ requests that RACF create a GENERAL audit record (event number 27). These checks are only carried out at connect time, so the performance cost is minimal.

You can report the IBM MQ general audit records using the RACF report writer (RACFRW). You could use the following RACFRW commands to report the RESLEVEL access:

```
RACFRW
SELECT PROCESS
EVENT GENERAL
LIST
END
```

A sample report from RACFRW, excluding the `Date`, `Time`, and `SYSID` fields, is shown in Figure 72.

```
       RACF REPORT - LISTING OF PROCESS RECORDS                                     PAGE   4
                                  E
                                  V  Q
                                  E  U
*JOB/USER *STEP/  --TERMINAL--  N  A
   NAME    GROUP     ID    LVL  T  L

 WS21B    MQMGRP IGJZM000   0   27 0  JOBID=(WS21B 05.111 09:44:57),USERDATA=()
    TRUSTED USER                      AUTH=(NONE),REASON=(NONE)
                                   SESSION=TSOLOGON,TERMINAL=IGJZM000,
                                   LOGSTR='CSQH RESLEVEL CHECK PERFORMED AGAINST PROFILE(QM66.RESLEVEL),
                                   CLASS(MQADMIN), ACCESS EQUATES TO (CONTROL)',RESULT=SUCCESS,MQADMIN
```

Figure 72. Sample output from RACFRW showing RESLEVEL general audit records

From checking the LOGSTR data in this sample output, you can see that TSO user WS21B has CONTROL access to QM66.RESLEVEL. This means that all resource security checks are bypassed when user WS21B access QM66 resources.

For more information about using RACFRW, see the *z/OS Security Server RACF Auditor's Guide*.

Customizing security

If you want to change the way IBM MQ security operates, you must do this through the SAF exit (ICHRFR00), or exits in your external security manager.

To find out more about RACF exits, see the *z/OS Security Server RACROUTE Macro Reference* manual.

Note: Because IBM MQ optimizes calls to the ESM, RACROUTE requests might not be made on, for example, every open for a particular queue by a particular user.

Security violation messages on z/OS

A security violation is indicated by the return code MQRC_NOT_AUTHORIZED in an application program or by a message in the job log.

A return code of MQRC_NOT_AUTHORIZED can be returned to an application program for the following reasons:

- A user is not allowed to connect to the queue manager. In this case, you get an ICH408I message in the Batch/TSO, CICS, or IMS job log.
- A user sign-on to the queue manager has failed because, for example, the job user ID is not valid or appropriate, or the task user ID or alternate user ID is not valid. One or more of these user IDs might not be valid because they have been revoked or deleted. In this case, you get an ICHxxxx message and possibly an IRRxxxx message in the queue manager job log giving the reason for the sign-on failure. For example:

  ```
  ICH408I USER(NOTDFND ) GROUP(        ) NAME(???                 )
    LOGON/JOB INITIATION - USER AT TERMINAL          NOT RACF-DEFINED
  IRR012I  VERIFICATION FAILED. USER PROFILE NOT FOUND
  ```
- An alternate user has been requested, but the job or task user ID does not have access to the alternate user ID. For this failure, you get a violation message in the job log of the relevant queue manager.
- A context option has been used or is implied by opening a transmission queue for output, but the job user ID or, where applicable, the task or alternate user ID does not have access to the context option. In this case, a violation message is put in the job log of the relevant queue manager.
- An unauthorized user has attempted to access a secured queue manager object, for example, a queue. In this case, an ICH408I message for the violation is put in the job log of the relevant queue manager. This violation might be due to the job or, when applicable, the task or alternate user ID.

Violation messages for command security and command resource security can also be found in the job log of the queue manager.

If the ICH408I violation message shows the queue manager jobname rather than a user ID, this is normally the result of a blank alternate user ID being specified. For example:

```
ICH408I JOB(MQS1MSTR) STEP(MQS1MSTR)
         MQS1.PAYROLL.REQUEST CL(MQQUEUE)
         INSUFFICIENT ACCESS AUTHORITY
         ACCESS INTENT(UPDATE )  ACCESS ALLOWED(NONE   )
```

You can find out who is allowed to use blank alternate user IDs by checking the access list of the MQADMIN profile hlq.ALTERNATE.USER.-BLANK-.

An ICH408I violation message can also be generated by:

- A command being sent to the system-command input queue without context. User-written programs that write to the system-command input queue should always use a context option. For more information, see "Profiles for context security" on page 598.

- When the job accessing the IBM MQ resource does not have a user ID associated with it, or when an IBM MQ adapter cannot extract the user ID from the adapter environment.

Violation messages might also be issued if you are using both queue-sharing group and queue manager level security. You might get messages indicating that no profile has been found at queue manager level, but still be granted access because of a queue-sharing group level profile.

```
ICH408I JOB(MQS1MSTR) STEP(MQS1MSTR)
         MQS1.PAYROLL.REQUEST CL(MQQUEUE)
         PROFILE NOT FOUND - REQUIRED FOR AUTHORITY CHECKING
         ACCESS INTENT(UPDATE )  ACCESS ALLOWED(NONE   )
```

What to do if access is allowed or disallowed incorrectly

In addition to the steps detailed in the *z/OS Security Server RACF Security Administrator's Guide*, use this checklist if access to a resource appears to be incorrectly controlled.

- Are the switch profiles correctly set?
  - Is RACF active?
  - Are the IBM MQ RACF classes installed and active?

    Use the RACF command, SETROPTS LIST, to check this.
  - Use the IBM MQ DISPLAY SECURITY command to display the current switch status from the queue manager.
  - Check the switch profiles in the MQADMIN class.

    Use the RACF commands, SEARCH and RLIST, for this.
  - Recheck the RACF switch profiles by issuing the IBM MQ REFRESH SECURITY(MQADMIN) command.
- Has the RACF resource profile changed? For example, has universal access on the profile changed or has the access list of the profile changed?
  - Is the profile generic?

    If it is, issue the RACF command, SETROPTS GENERIC(classname) REFRESH.
  - Have you refreshed the security on this queue manager?

    If required, issue the RACF command SETROPTS RACLIST(classname) REFRESH.

    If required, issue the IBM MQ REFRESH SECURITY(*) command.
- Has the RACF definition of the user changed? For example, has the user been connected to a new group or has the user access authority been revoked?
  - Have you reverified the user by issuing the IBM MQ RVERIFY SECURITY(userid) command?
- Are security checks being bypassed due to RESLEVEL?
  - Check the connecting user ID's access to the RESLEVEL profile. Use the RACF audit records to determine what the RESLEVEL is set to.
  - For channels, remember that the access level that the channel initiator's userid has to RESLEVEL is inherited by all channels, so an access level, such as ALTER, that causes all checks to be bypassed causes security checks to be bypassed for all channels.
  - If you are running from CICS, check the transaction's RESSEC setting.
  - If RESLEVEL has been changed while a user is connected, they must disconnect and reconnect before the new RESLEVEL setting takes effect.
- Are you using queue-sharing groups?
  - If you are using both queue-sharing group and queue manager level security, check that you have defined all the correct profiles. If queue manager profile is not defined, a message is sent to the log stating that the profile was not found.

– Have you used a combination of switch settings that is not valid so that full security checking has been set on?
– Do you need to define security switches to override some of the queue-sharing group settings for your queue manager?
– Is a queue manager level profile taking precedence over a queue-sharing group level profile?

Security considerations for the channel initiator on z/OS

If you are using resource security in a distributed queuing environment, the Channel initiator address space needs appropriate access to various IBM MQ resources. You can use the Integrated Cryptographic Support Facility (ICSF) to seed the password protection algorithm.

Using resource security

If you are using resource security, consider the following points if you are using distributed queuing:

System queues
The channel initiator address space needs RACF UPDATE access to the system queues listed at "System queue security" on page 588, and to all the user destination queues and the dead-letter queue (but see "Dead-letter queue security" on page 587).

Transmission queues
The channel initiator address space needs ALTER access to all the user transmission queues.

Context security
The channel user ID (and the MCA user ID if one has been specified) need RACF CONTROL access to the hlq.CONTEXT.queuename profiles in the MQADMIN class. Depending on the RESLEVEL profile, the channel user ID might also need CONTROL access to these profiles.

All channels need CONTROL access to the MQADMIN hlq.CONTEXT. dead-letter-queue profile. All channels (whether initiating or responding) can generate reports, and consequently they need CONTROL access to the hlq.CONTEXT.reply-q profile.

SENDER, CLUSSDR, and SERVER channels need CONTROL access to the hlq.CONTEXT.xmit-queue-name profiles since messages can be put onto the transmission queue to wake up the channel to end gracefully.

Note: If the channel user ID, or a RACF group to which the channel user ID is connected, has CONTROL or ALTER access to the hlq.RESLEVEL, then there are no resource checks for the channel initiator or any of its channels.

See "Profiles for context security" on page 598 "RESLEVEL and the channel initiator connection" on page 615 and "User IDs for security checking" on page 617 for more information.

CSQINPX
If you are using the CSQINPX input data set, the channel initiator also needs READ access to CSQINPX, and UPDATE access to data set CSQOUTX and dynamic queues SYSTEM.CSQXCMD.*.

Connection security
The channel initiator address space connection requests use a connection type of CHIN, for which appropriate access security must be set, see "Connection security profiles for the channel initiator" on page 581.

Data sets
The channel initiator address space needs appropriate access to queue manager data sets, see "Authorizing access to data sets" on page 633.

Commands

The distributed queuing commands (for example, DEFINE CHANNEL, START CHINIT, START LISTENER, and other channel commands) must have appropriate command security set, see Table 59 on page 601.

If you are using a queue-sharing group, the channel initiator might issue various commands internally, so the user ID it uses must be authorized to issue such commands. These commands are START and STOP CHANNEL for every channel used with CHLDISP(SHARED).

If the PSMODE of the queue manager is not DISABLED, the channel initiator must have READ access to the DISPLAY PUBSUB command.

Channel security

Channels, particularly receivers and server-connections, need appropriate security to be set up; see "User IDs for security checking" on page 617 for more information.

You can also use the Transport Layer Security (TLS) or Secure Sockets Layer (SSL) protocols to provide security on channels. See "SSL and TLS security protocols in IBM MQ" on page 397 for more information about using TLS and SSL with IBM MQ.

See also "Access control for clients" on page 467 for information about server-connection security.

User IDs

The user IDs described in "User IDs used by the channel initiator" on page 620 and "User IDs used by the intra-group queuing agent" on page 624 need the following access:

- RACF UPDATE access to the appropriate destination queues and the dead-letter queue
- RACF CONTROL access to the `hlq.CONTEXT.queuename` profile if context checking is performed at the receiver
- Appropriate access to the hlq.ALTERNATE.USER.userid profiles they might need to use.
- For clients, the appropriate RACF access to the resources to be used.

APPC security

Set appropriate APPC security if you are using the LU 6.2 transmission protocol. (Use the APPCLU RACF class for example.) For information about setting up security for APPC, see the following manuals:

- *z/OS V1R2.0 MVS Planning: APPC Management*
- *Multiplatform APPC Configuration Guide*, an IBM Redbooks publication

Outbound transmissions use the "SECURITY(SAME)" APPC option. As a result, the user ID of the channel initiator address space and its default profile (RACF GROUP) are flowed across the network to the receiver with an indicator that the user ID has already been verified (ALREADYV).

If the receiving side is also z/OS, the user ID and profile are verified by APPC and the user ID is presented to the receiver channel and used as the channel user ID.

In an environment where the queue manager is using APPC to communicate with another queue manager on the same or another z/OS system, you need to ensure that either:

- The VTAM definition for the communicating LU specifies SETACPT(ALREADYV)
- There is a RACF APPCLU profile for the connection between LUs that specifies CONVSEC(ALREADYV)

Changing security settings

If the RACF access level that either the channel user ID or MCA user ID has to a destination queue is changed, this change takes effect only for new object handles (that is, new **MQOPEN** s) for the destination queue. The times when MCAs open and close queues is variable; if a channel is already running when such an access change is made, the MCA can continue to put messages on the destination queue using the existing security access of the user IDs rather than the updated security access. Stopping and restarting the channels to enforce the updated access level avoids this scenario.

Automatic restart
: If you are using the z/OS Automatic Restart Manager (ARM) to restart the channel initiator, the user ID associated with the XCFAS address space must be authorized to issue the IBM MQ START CHINIT command.

Using the Integrated Cryptographic Service Facility (ICSF)

The channel initiator can use ICSF to generate a random number when seeding the password protection algorithm to obfuscate passwords flowing over client channels if SSL/TLS is not being used. The process of generating a random number is called *entropy*.

If you have the z/OS feature installed but have not started ICSF, you see message CSQX213E and the channel initiator uses STCK for entropy.

Message CSQX213E warns you that the password protection algorithm is not as secure as it could be. However, you can continue your process; there is no other impact on runtime.

If you do not have the z/OS feature installed, the channel initiator automatically uses STCK.

Notes:
1. Using ICSF for entropy generates more random sequences than using STCK.
2. If you start ICSF you must restart the channel initiator.
3. ICSF is required for certain CipherSpecs. If you attempt to use one of these CipherSpecs and you do not have ICSF installed, you receive message CSQX629E.

Security in queue manager clusters on z/OS

Security considerations for clusters are the same for queue managers and channels that are not clustered. The channel initiator needs access to some additional system queues, and some additional commands need appropriate security set.

You can use the MCA user ID, channel authentication records, SSL or TLS, and security exits to authenticate cluster channels (as with conventional channels). The channel authentication records or security exit relating to the cluster-receiver channel must check that the remote queue manager is permitted access to the server queue manager's cluster queues. You can start to use IBM MQ cluster support without changing your existing queue access security. You must, however, allow other queue managers in the cluster to write to the SYSTEM.CLUSTER.COMMAND.QUEUE if they are to join the cluster.

IBM MQ cluster support does not provide a mechanism to limit a member of a cluster to the client role only. As a result, you must be sure that you trust any queue managers that you allow into the cluster. If any queue manager in the cluster creates a queue with a particular name, it can receive messages for that queue, regardless of whether the application putting messages to that queue intended this or not.

To restrict the membership of a cluster, take the same action that you would take to prevent queue managers connecting to receiver channels. You restrict the membership of a cluster by using channel authentication records or by writing a security exit program on the receiver channel. You can also write an exit program to prevent unauthorized queue managers from writing to the SYSTEM.CLUSTER.COMMAND.QUEUE.

Note: It is not advisable to permit applications to open the SYSTEM.CLUSTER.TRANSMIT.QUEUE directly. It is also not advisable to permit an application to open any other transmission queue directly.

If you are using resource security, consider the following points in addition to the considerations contained in “Security considerations for the channel initiator on z/OS” on page 641:

System queues
The channel initiator needs RACF ALTER access to the following system queues:
- SYSTEM.CLUSTER.COMMAND QUEUE
- SYSTEM.CLUSTER.TRANSMIT.QUEUE.

and UPDATE access to SYSTEM.CLUSTER.REPOSITORY.QUEUE

It also needs READ access to any namelists used for clustering.

Commands
Set appropriate command security (as described in Table 59 on page 601) for the cluster support commands (REFRESH and RESET CLUSTER, SUSPEND, and RESUME QMGR.

Security considerations for using IBM MQ with CICS

The CICS adapter provides information to IBM MQ for use in security.

The CICS adapter provides the following information to IBM MQ specifically for use in IBM MQ security:
- Whether CICS resource-level security is active for this transaction, as specified on the RESSEC or RSLC operand of the RDO TRANSACTION definition.
- User IDs.

  For terminal tasks where a user has not signed on, the user ID is the CICS user ID associated with the terminal and is either:
  - The default CICS user ID as specified on the CICS parameter DFLTUSER SIT
  - A preset security user ID specified on the terminal definition

  For non-terminal tasks, the CICS adapter tries to get a user ID with an EXEC CICS ASSIGN command. If this is unsuccessful, the adapter tries to get the user ID using EXEC CICS INQUIRE TASK. If security is active in CICS, and the non-terminal attached transaction is defined with CMDSEC(YES), the CICS adapter passes a user ID of blanks to IBM MQ.

For details of security considerations, see:
- Security for the CICS-IBM MQ adapter.
- Security for the CICS-IBM MQ bridge.

Controlling the security of CICS transactions supplied by IBM MQ

If you want a user to administer the CICS adapter, grant the user authorization to certain transactions.

The CKTI and CKAM transactions are designed to be run without a terminal; no user should have access to these transactions. These transactions are examples of what the *CICS RACF Security Guide* calls "category 2 transactions". For information about how to set up these transactions in CICS and RACF, see the information about category 2 transactions in the *CICS RACF Security Guide*.

If you want a user to administer the CICS adapter, you must grant the user authorization to these transactions:

CKQC	Controls the CICS adapter functions
CKBM	Controls the CICS adapter functions
CKRT	Controls the CICS adapter functions
CKCN	Connect
CKSD	Disconnect
CKRS	Statistics
CKDP	Full screen display
CKDL	Line mode display
CKSQ	CKTI START/STOP

If required, you can restrict access to specific functions of the adapter. For example, if you want to allow users to display the status of the adapter through the full screen interface, but nothing else, give them access to CKQC, CKBM, CKRT, and CKDP only.

Define these transactions to CICS with RESSEC(NO) and CMDSEC(NO). For more details, see the *CICS RACF Security Guide*.

The CICS adapter user ID

The user ID associated with the CICS adapter is that of the IBM MQ-supplied task initiator transaction, CKTI. This has a number of implications.

User ID checking for IBM MQ resources during PLTPI and PLTSD

If an IBM MQ resource is accessed during the CICS PLTPI phase, the user ID passed to IBM MQ is blanks. If an IBM MQ resource is accessed during the CICS PLTSD phase, the user ID passed to IBM MQ is the user ID associated with the shutdown transaction.

If CKTI is started during the CICS PLTPI phase, the user ID of the CKTI task is the CICS sysidnt. This means that a user ID with the same name as the CICS sysidnt must be defined and given access to the required IBM MQ resources, for example, initiation queues.

Terminal user IDs

If CKTI is started from a terminal from the CKQC transaction or a user-written program that links to CSQCSSQ, the user ID that CKTI uses is the same as the user ID of the terminal that started CKTI.

Automating starting of CKTI

To automate the starting of CKTIs under a specific user ID, you can use an automation product, for example, Tivoli® NetView® for z/OS. You can use this to sign on a CICS console and issue the STARTCKTI command.

You can also use preset security sequential terminals, which have been defined to emulate a CRLP terminal, with the sequential terminal input containing the CKQC STARTCKTI command.

However, when the CICS adapter alert monitor reconnects CICS to IBM MQ, after, for example, an IBM MQ restart, only the CKTI specified at the initial IBM MQ connection is restarted. You must automate starting any extra CKTIs yourself.

Propagating the CKTI user ID to other CICS transactions

If CKTI starts other CICS transactions, for example, message channel agents (MCAs) or user-written CICS applications, the user ID of CKTI is propagated to these applications. For example, if CKTI is running under user ID *USERNAME* and a trigger event occurs that requires the sender MCA transaction, *TRNA*, to be started, the *TRNA* transaction also runs under user ID *USERNAME*. Therefore user ID *USERNAME* must have access to the required transmission queue.

Security considerations for the CICS bridge

When you run the CICS bridge, you can specify the level of authentication you want to take place.

If requested, the bridge checks the user ID and password extracted from the IBM MQ request message before running the CICS program named in the request message. The queue manager uses the external security manager (ESM) (for example RACF) to do authentication. Therefore, user IDs in the request message must be defined to the ESM.

Note:

1. If you have not specified a user ID in the message descriptor (MQMD) or password in the CICS bridge header (MQCIH) of a message, the bridge task runs with the LOCAL level of authentication, even if you started the bridge monitor with a different authentication option.
2. The options that include password (or PassTicket) validation require an MQCIH to be provided. See MQCIH - CICS bridge header for more information about the MQCIH header.
3. PassTicket validation is performed using IBM MQ services, not EXEC CICS VERIFY, as the CICS service does not accept an APPLID.

The level of authentication you can use is described using the following keywords:

LOCAL
This is the default value. CICS programs run by the bridge task are started with the CICS DFLTUSER user ID, therefore run with the authority associated with this user ID. There is no checking of user IDs or passwords. If a CICS program is run that tries to access protected resources, it is likely to fail.

IDENTIFY
When you start the monitor task with the IDENTIFY authentication option, the bridge task is started with the user ID specified in the message (MQMD). CICS programs run by the bridge run with the user ID from the MQMD. There is no password checking, the user ID is treated as trusted.

VERIFY_UOW
If MQMD.PutApplType is set to MQAT_NO_CONTEXT, the CICS DFLTUSER user ID is used, as for the LOCAL authentication option. Otherwise, the bridge monitor checks the user ID (in the MQMD) and password (in the CIH) before starting the bridge task, and CICS programs run by the bridge run with the user ID extracted from the MQMD. If the user ID or password is invalid, the request fails with return code MQCRC_SECURITY_ERROR. Subsequent messages processed by this transaction are not checked.

Bridge tasks that cannot be started because of a security failure are tried again if ROUTEMEM=N (the default) is specified. However, if ROUTEMEM=Y is specified the bridge messages are put to the dead-letter queue.

VERIFY_ALL
This value is the same as VERIFY_UOW except that the bridge task checks the user ID and password in every message. This option is not applicable for 3270 transactions when using CICS earlier than CICS Transaction Server Version 2 Release 2.

A PassTicket can be used in place of a password to avoid the need to flow passwords in messages (see Security Server RACF System Programmer's Guide). When generating a PassTicket an APPLID must be specified. If you are using a single bridge monitor, the APPLID is the CICS APPLID unless a different value was specified when the bridge was started. If you are using multiple bridge monitors for a queue, you must specify the APPLID to be used, using the PASSTKTA= *applid* parameter at bridge startup.

If you have not specified a user ID in a message, or you have not provided a password, the CICS program started by the CICS bridge runs with the user ID set to the user ID used to start the bridge monitor, regardless of the option requested. If you want more than one level of authentication checking performed, run a monitor task for each level you need.

When a CICS DPL request is read by the bridge monitor it starts the transaction specified in the CICS bridge header (MQCIH) or, if no transaction is specified, transaction CKBP. The user IDs under which the bridge monitor runs must have authority to start the various transactions that might be requested. The default transaction ID for the CICS bridge monitor is CKBR but you can change this or define additional transaction IDs if you want more granular access to queues and transactions. You can use CICS surrogate security to restrict which user ID and transaction combinations a bridge monitor transaction and user ID can start.

Table 76 and Table 77 summarize the level of authority of the bridge monitor and the bridge tasks, and the use of the MQMD user ID.

Table 76. CICS bridge monitor security

Monitor started by	**At a signed on terminal**	**Monitor authority**
From a terminal or EXEC CICS LINK within a program	Yes	Signed on user ID
From a terminal or EXEC CICS LINK within a program	No	CICS default user ID
EXEC CICS START with user ID	-	User ID from START
EXEC CICS START without user ID	-	CICS default user ID
The IBM MQ trigger monitor CKTI	-	CICS default user ID

Table 77. CICS bridge task security

AUTH	**Bridge task authority**
LOCAL	CICS default user ID
IDENTIFY	MQMD UserIdentifier
VERIFY_UOW	MQMD UserIdentifier
VERIFY_ALL	MQMD UserIdentifier

The options IDENTIFY, VERIFY_UOW, and VERIFY_ALL need the user ID of the bridge monitor defined to RACF as a surrogate of all the user IDs used in request messages. This requirement is in addition to the user ID in the message being defined to RACF. (A surrogate user is one who has the authority to start work on behalf of another user, without knowing the other user's password.)

For more information about surrogate user security, see the *CICS RACF Security Guide*.

Note: When IDENTIFY security is being used, you might see abend AICO for CKBP if you try to run with a user ID that has been revoked. The error reply has return code MQCRC_BRIDGE_ERROR with reason MQFB_CICS_BRIDGE_FAILURE.

Authority for the CICS bridge:

Components of the bridge need authority to either put to or get from the various IBM MQ queues.

In summary, the required authority is as follows:
- The monitor and all bridge tasks need authority to get messages from the bridge request queue.
- A bridge task needs authority to put messages to its own reply-to queue.
- To ensure that any error replies are received, the monitor needs authority to put messages to all reply-to queues.
- Bridge tasks need authority to put messages to the dead-letter queue.
- The monitor needs authority to put messages to the dead-letter queue, unless you want the bridge to stop if an error occurs.
- The monitor and all bridge tasks need authority to put messages to the backout requeue queue, if one is defined

See Table 76 on page 647 to determine the correlation between user IDs and authority.

Security considerations for using IBM MQ with IMS

Use this topic to plan your security requirements when you use IBM MQ with IMS.

Using the OPERCMDS class

If you are using RACF to protect resources in the OPERCMDS class, ensure that the userid associated with your IBM MQ queue manager address space has authority to issue the MODIFY command to any IMS system to which it can connect.

Security considerations for the IMS bridge

There are four aspects that you must consider when deciding your security requirements for the IMS bridge, these are:
- What security authorization is needed to connect IBM MQ to IMS
- How much security checking is performed on applications using the bridge to access IMS
- Which IMS resources these applications are allowed to use
- What authority is to be used for messages that are put and got by the bridge

When you define your security requirements for the IMS bridge you must consider the following:
- Messages passing across the bridge might have originated from applications on platforms that do not offer strong security features
- Messages passing across the bridge might have originated from applications that are not controlled by the same enterprise or organization

Security considerations for connecting to IMS

Grant the user ID of the IBM MQ queue manager address space access to the OTMA group.

The IMS bridge is an OTMA client. The connection to IMS operates under the user ID of the IBM MQ queue manager address space. This is normally defined as a member of the started task group. This user ID must be granted access to the OTMA group (unless the /SECURE OTMA setting is NONE).

To do this, define the following profile in the FACILITY class:

```
IMSXCF.xcfgname.mqxcfmname
```

Where `xcfgname` is the XCF group name and `mqxcfmname` is the XCF member name of IBM MQ.

You must give your IBM MQ queue manager user ID read access to this profile.

Note:

1. If you change the authorities in the FACILITY class, you must issue the RACF command SETROPTS RACLIST(FACILITY) REFRESH to activate the changes.
2. If profile hlq.NO.SUBSYS.SECURITY exists in the MQADMIN class, no user ID is passed to IMS and the connection fails unless the /SECURE OTMA setting is NONE.

Application access control for the IMS bridge

Define a RACF profile in the FACILITY class for each IMS system. Grant an appropriate level of access to the IBM MQ queue manager user ID.

For each IMS system that the IMS bridge connects to, you can define the following RACF profile in the FACILITY class to determine how much security checking is performed for each message passed to the IMS system.

```
IMSXCF.xcfgname.imsxcfmname
```

Where `xcfgname` is the XCF group name and `imsxcfmname` is the XCF member name for IMS. (You need to define a separate profile for each IMS system.)

The access level you allow for the IBM MQ queue manager user ID in this profile is returned to IBM MQ when the IMS bridge connects to IMS, and indicates the level of security that is required on subsequent transactions. For subsequent transactions, IBM MQ requests the appropriate services from RACF and, where the user ID is authorized, passes the message to IMS.

OTMA does not support the IMS /SIGN command; however, IBM MQ allows you to set the access checking for each message to enable implementation of the necessary level of control.

The following access level information can be returned:

NONE or NO PROFILE FOUND
These values indicate that maximum security is required, that is, authentication is required for every transaction. A check is made to verify that the user ID specified in the *UserIdentifier* field of the MQMD structure, and the password or PassTicket in the *Authenticator* field of the MQIIH structure are known to RACF, and are a valid combination. A UTOKEN is created with a password or PassTicket, and passed to IMS ; the UTOKEN is not cached.

Note: If profile hlq.NO.SUBSYS.SECURITY exists in the MQADMIN class, this level of security overrides whatever is defined in the profile.

READ This value indicates that the same authentication is to be performed as for `NONE` under the following circumstances:

- The first time that a specific user ID is encountered
- When the user ID has been encountered before but the cached UTOKEN was not created with a password or PassTicket

IBM MQ requests a UTOKEN if required, and passes it to IMS.

Note: If a request to reverify security has been acted on, all cached information is lost and a UTOKEN is requested the first time each user ID is later encountered.

UPDATE
A check is made that the user ID in the *UserIdentifier* field of the MQMD structure is known to RACF.

A UTOKEN is built and passed to IMS ; the UTOKEN is cached.

CONTROL/ALTER
These values indicate that no security UTOKENs need to be provided for any user IDs for this IMS system. (You would probably only use this option for development and test systems.)

Attention: Note that the user ID contained in the *UserIdentifier* field of the MQMD structure is still passed for **`CONTROL/ALTER`**.

Note:
1. This access is defined when IBM MQ connects to IMS, and lasts for the duration of the connection. To change the security level, the access to the security profile must be changed and then the bridge stopped and restarted (for example, by stopping and restarting OTMA).
2. If you change the authorities in the FACILITY class, you must issue the RACF command SETROPTS RACLIST(FACILITY) REFRESH to activate the changes.
3. You can use a password or a PassTicket, but you must remember that the IMS bridge does not encrypt data. For information about using PassTickets, see "Using RACF PassTickets in the IMS header" on page 652.
4. Some of these results might be affected by security settings in IMS, using the /SECURE OTMA command.
5. Cached UTOKEN information is held for the duration defined by the INTERVAL and TIMEOUT parameters of the IBM MQ ALTER SECURITY command.
6. The RACF WARNING option has no effect on the IMSXCF.xcfgname.imsxcfmname profile. Its use does not affect the level of access granted, and no RACF WARNING messages are produced.

Security checking on IMS

Messages that pass across the bridge contain security information. The security checks made depend on the setting of the IMS command /SECURE OTMA.

Each IBM MQ message that passes across the bridge contains the following security information:
- A user ID contained in the *UserIdentifier* field of the MQMD structure
- The security scope contained in the *SecurityScope* field of the MQIIH structure (if the MQIIH structure is present)
- A UTOKEN (unless the IBM MQ sub system has CONTROL or ALTER access to the relevant `IMSXCF.xcfgname.imsxcfmname` profile)

The security checks made depend on the setting of the IMS command /SECURE OTMA, as follows:

/SECURE OTMA NONE
No security checks are made for the transaction.

/SECURE OTMA CHECK
The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking.

An ACEE (Accessor Environment Element) is built in the IMS control region.

/SECURE OTMA FULL
The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking.

An ACEE is built in the IMS dependent region as well as the IMS control region.

/SECURE OTMA PROFILE
The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking

The *SecurityScope* field in the MQIIH structure is used to determine whether to build an ACEE in the IMS dependent region as well as the control region.

Note:

1. If you change the authorities in the TIMS or CIMS class, or the associated group classes GIMS or DIMS, you must issue the following IMS commands to activate the changes:
   - /MODIFY PREPARE RACF
   - /MODIFY COMMIT
2. If you do not use /SECURE OTMA PROFILE, any value specified in the *SecurityScope* field of the MQIIH structure is ignored.

Security checking done by the IMS bridge

Different authorities are used depending on the action being performed.

When the bridge puts or gets a message, the following authorities are used:

Getting a message from the bridge queue
No security checks are performed.

Putting an exception, or COA report message
Uses the authority of the user ID in the *UserIdentifier* field of the MQMD structure.

Putting a reply message
Uses the authority of the user ID in the *UserIdentifier* field of the MQMD structure of the original message

Putting a message to the dead-letter queue
No security checks are performed.

Note:

1. If you change the IBM MQ class profiles, you must issue the IBM MQ REFRESH SECURITY(*) command to activate the changes.
2. If you change the authority of a user, you must issue the MQSC RVERIFY SECURITY command to activate the change.

Using RACF PassTickets in the IMS header

You can use a PassTicket in place of a password in the IMS header.

If you want to use a PassTicket instead of a password in the IMS header (MQIIH), specify the application name against which the PassTicket is validated in the PASSTKTA attribute of the STGCLASS definition of the IMS bridge queue to which the message is to be routed.

If the PASSTKTA value is left blank, you must arrange to have a PassTicket generated. The application name in this case must be of the form MVSxxxx, where xxxx is the SMFID of the z/OS system on which the target queue manager runs.

A PassTicket is built from a user ID, the target application name, and a secret key. It is an 8-byte value containing uppercase alphabetic and numeric characters. It can be used only once, and is valid for a 20 minute period. If a PassTicket is generated by a local RACF system, RACF only checks that the profile exists and not that the user has authority against the profile. If the PassTicket was generated on a remote system, RACF validates the access of the user ID to the profile. For full information about PassTickets, see the *z/OS SecureWay Security Server RACF Security Administrator's Guide*.

PassTickets in IMS headers are given to RACF by IBM MQ, not IMS.

Setting up IBM MQ MQI client security

You must consider IBM MQ MQI client security, so that the client applications do not have unrestricted access to resources on the server.

When running a client application, do not run the application using a user ID that has more access rights than necessary; for example, a user in the `mqm` group or even the `mqm` user itself.

By running an application as a user with too many access rights, you run the risk of the application accessing and changing parts of the queue manager, either by accident or maliciously.

There are two aspects to security between a client application and its queue manager server: authentication and access control.

- Authentication can be used to ensure that the client application, running as a specific user, is who they say they are. By using authentication you can prevent an attacker from gaining access to your queue manager by impersonating one of your applications.

  From IBM MQ Version 8.0, authentication is provided by one of two options:
  - The connection authentication feature.

    For more information on connection authentication, see "Connection authentication" on page 430.
  - Using mutual authentication within SSL or TLS.

    For more information on SSL or TLS, see "Working with SSL or TLS" on page 658.
- Access control can be used to give or remove access rights for a specific user or group of users. By running a client application with a specifically created user (or user in a specific group) you can then use access controls to ensure the application cannot access parts of your queue manager that the application is not supposed to.

  When setting up access control you must consider channel authentication rules and the MCAUSER field on a channel. Both of these features have the ability to change which user id is being used for verifying access control rights.

  For more information on access control, see "Authorizing access to objects" on page 735.

If you have set up a client application to connect to a specific channel with a restricted ID, but the channel has an administrator ID set in its MCAUSER field then, provided the client application connects successfully, the administrator ID is used for access control checks. Therefore, the client application will have full access rights to your queue manager.

For more information on the MCAUSER attribute, see “Mapping a client asserted user ID to an MCAUSER user ID” on page 765.

Channel authentication rules can also be used as a method for controlling access to a queue manager, by setting up specific rules and criteria for a connection to be accepted.

For more information on channel authentication rules see: “Channel authentication records” on page 424.

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

In order to be FIPS-compliant at run time, the key repositories must have been created and managed using only FIPS-compliant software such as runmqakm with the -fips option.

You can specify that an SSL or TLS channel must use only FIPS-certified CipherSpecs in three ways, listed in order of precedence:

1. Set the FipsRequired field in the MQSCO structure to MQSSL_FIPS_YES.
2. Set the environment variable MQSSLFIPS to YES.
3. Set the SSLFipsRequired attribute in the client configuration file to YES.

By default, FIPS-certified CipherSpecs is not required.

These values have the same meanings as the equivalent parameter values on ALTER QMGR SSLFIPS (see ALTER QMGR). If the client process currently has no active SSL or TLS connections, and a FipsRequired value is validly specified on an SSL MQCONNX, all subsequent SSL connections associated with this process must use only the CipherSpecs associated with this value. This applies until this and all other SSL or TLS connections have stopped, at which stage a subsequent MQCONNX can provide a new value for FipsRequired.

If cryptographic hardware is present, the cryptographic modules used by IBM MQ can be configured to be those modules provided by the hardware product, and these might be FIPS-certified to a particular level. The configurable modules and whether they are FIPS-certified depends on the hardware product in use.

Where possible, if FIPS-only CipherSpecs is configured then the MQI client rejects connections which specify a non-FIPS CipherSpec with MQRC_SSL_INITIALIZATION_ERROR. IBM MQ does not guarantee to reject all such connections and it is your responsibility to determine whether your IBM MQ configuration is FIPS-compliant.

Related concepts:

"Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows" on page 407
When cryptography is required on an SSL or TLS channel on Windows, UNIX and Linux systems, IBM MQ uses a cryptography package called IBM Crypto for C (ICC). On the Windows, UNIX and Linux platforms, the ICC software has passed the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2.

Related information:

FipsRequired (MQLONG)

MQSSLFIPS

SSL stanza of the client configuration file

Running SSL or TLS client applications with multiple installations of GSKit V8.0 on AIX

SSL or TLS client applications on AIX might experience `MQRC_CHANNEL_CONFIG_ERROR` and error `AMQ6175` when running on AIX systems with multiple GSKit V8.0 installations.

When running client applications on an AIX system with multiple GSKit V8.0 installations, the client connect calls can return `MQRC_CHANNEL_CONFIG_ERROR` when using SSL or TLS. The `/var/mqm/errors` logs record error `AMQ6175` and `AMQ9220` for the failing client application, for example:

```
09/08/11 11:16:13 - Process(24412.1) User(user) Program(example)
                    Host(machine.example.ibm.com) Installation(Installation1)
                    VRMF(7.1.0.0)
AMQ6175: The system could not dynamically load the shared library
'/usr/mqm/gskit8/lib64/libgsk8ssl_64.so'. The system returned
error number '8' and error message 'Symbol resolution failed
for /usr/mqm/gskit8/lib64/libgsk8ssl_64.so because:
    Symbol VALUE_EC_NamedCurve_secp256r1__9GSKASNOID (number 16) is not
exported from dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
    Symbol VALUE_EC_NamedCurve_secp384r1__9GSKASNOID (number 17) is not exported from
dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
    Symbol VALUE_EC_NamedCurve_secp521r1__9GSKASNOID (number 18) is not exported from
dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
    Symbol VALUE_EC_ecPublicKey__9GSKASNOID (number 19) is not exported from dependent
module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
    Symbol VALUE_EC_ecdsa_with_SHA1__9GSKASNOID (number 20) is not exported from
dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
    Symbol VALUE_EC_ecdsa__9GSKASNOID (number 21) is not exported from dependent
      module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.'.

EXPLANATION:
This message applies to AIX systems. The shared library
'/usr/mqm/gskit8/lib64/libgsk8ssl_64.so' failed
to load correctly due to a problem with the library.
ACTION:
Check the file access permissions and that the file has not been corrupted.
----- amqxufnx.c : 1284 -------------------------------------------------------
09/08/11 11:16:13 - Process(24412.1) User(user) Program(example)
                    Host(machine.example.ibm.com) Installation(Installation1)
                    VRMF(7.1.0.0)
AMQ9220: The GSKit communications program could not be loaded.

EXPLANATION:
The attempt to load the GSKit library or procedure
'/usr/mqm/gskit8/lib64/libgsk8ssl_64.so' failed with error code
536895861.
ACTION:
Either the library must be installed on the system or the environment changed
to allow the program to locate it.
----- amqcgska.c : 836 --------------------------------------------------------
```

A common cause of this error is that the setting of the LIBPATH or LD_LIBRARY_PATH environment variable has caused the IBM MQ client to load a mixed set of libraries from two different GSKit V8.0 installations. Executing an IBM MQ client application in a Db2 environment can cause this error.

To avoid this error, include the IBM MQ library directories at the front of the library path so that the IBM MQ libraries take precedence. This can be achieved using the **setmqenv** command with the **-k** parameter, for example:

```
. /usr/mqm/bin/setmqenv -s -k
```

For more information about the use of the **setmqenv** command, refer to setmqenv (set IBM MQ environment)

Related information:

GSKit: Changes from GSKit V7.0 to GSKit V8.0

GSKit: Commands renamed

Setting up communications for SSL or TLS on IBM i

IBM i

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.

To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also create and manage your digital certificates. On some operating systems, you can perform the tests with self–signed certificates. However, on IBM i, you must use personal certificates signed by a local CA.

For full information about creating and managing certificates, see “Working with SSL or TLS on IBM i” on page 661.

This collection of topics introduces some of the tasks involved in setting up SSL or TLS communications, and provides step-by-step guidance on completing those tasks

You might also want to test SSL or TLS client authentication, which are optional parts of the SSL and TLS protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

On IBM i, the SSL or TLS client sends a certificate only if it has one labeled in the correct IBM MQ format:

- For a queue manager, ibmwebspheremq followed by the name of your queue manager changed to lower case. For example, for QM1, ibmwebspheremqqm1.
- For an IBM MQ C Client for IBM i, ibmwebspheremq followed by your logon user ID changed to lower case, for example ibmwebspheremqmyuserid.

IBM MQ uses the ibmwebspheremq prefix on a label to avoid confusion with certificates for other products. Ensure that you specify the entire certificate label in lower case.

The SSL or TLS server always validates the client certificate if one is sent. If the SSL or TLS client does not send a certificate, authentication fails only if the end of the channel acting as the SSL or TLS server is defined with either the SSLCAUTH parameter set to REQUIRED or an SSLPEER parameter value set. For more information, see Connecting two queue managers using SSL or TLS.

Setting up communications for SSL or TLS on UNIX, Linux or Windows systems

distributed

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.

To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also create and manage your digital certificates. On UNIX, Linux and Windows systems, you can perform the tests with self–signed certificates.

Self-signed certificates cannot be revoked, which could allow an attacker to spoof an identity after a private key has been compromised. CAs can revoke a compromised certificate, which prevents its further use. CA-signed certificates are therefore safer to use in a production environment, though self-signed certificates are more convenient for a test system.

For full information about creating and managing certificates, see "Working with SSL or TLS on UNIX, Linux and Windows systems" on page 673.

This collection of topics introduces some of the tasks involved in setting up SSL communications, and provides step-by-step guidance on completing those tasks.

You might also want to test SSL oro TLS client authentication, which are an optional part of the protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

On UNIX, Linux and Windows systems, the SSL or TLS client sends a certificate only if it has one labeled in the correct IBM MQ format:

- For a queue manager, the format is `ibmwebspheremq` followed by the name of your queue manager changed to lower case. For example, for `QM1`, `ibmwebspheremqqm1`
- For an IBM MQ client, `ibmwebspheremq` followed by your logon user ID changed to lower case, for example `ibmwebspheremqmyuserid`.

IBM MQ uses the `ibmwebspheremq` prefix on a label to avoid confusion with certificates for other products. Ensure that you specify the entire certificate label in lower case.

The SSL or TLS server always validates the client certificate if one is sent. If the client does not send a certificate, authentication fails only if the end of the channel acting as the SSL or TLS server is defined with either the SSLCAUTH parameter set to REQUIRED or an SSLPEER parameter value set. For more information, see Connecting two queue managers using SSL or TLS.

Setting up communications for SSL or TLS on z/OS

z/OS

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.

To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also create and manage your digital certificates. On z/OS you can perform the tests with self-signed certificates, or with personal certificates signed by a local certificate authority (CA).

Self-signed certificates cannot be revoked, which could allow an attacker to spoof an identity after a private key has been compromised. CAs can revoke a compromised certificate, which prevents its further use. CA-signed certificates are therefore safer to use in a production environment, though self-signed certificates are more convenient for a test system.

For full information about creating and managing certificates, see “Working with SSL or TLS on z/OS” on page 703.

This collection of topics introduces some of the tasks involved in setting up SSL or TLS communications, and provides step-by-step guidance on completing those tasks.

You might also want to test SSL or TLS client authentication, which are an optional part of the protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

On z/OS the SSL or TLS client sends a certificate only if it has one of the following certificates:

- For a shared channel only, a certificate with a label in the format `ibmWebSphereMQ` followed by the name of your queue-sharing group, for example `ibmWebSphereMQQSG1`
- A certificate with a label in the format`ibmWebSphereMQ` followed by the name of your queue manager, for example `ibmWebSphereMQQM1`
- A default certificate (which might be the `ibmWebSphereMQ` certificate).

If the channel is shared, the channel first tries to find a certificate for the queue-sharing group. If it does not find a certificate for a queue-sharing group, it tries to find a certificate for the queue manager.

On z/OS, IBM MQ uses the `ibmWebSphereMQ` prefix on a label to avoid confusion with certificates for other products.

The SSL or TLS server always validates the client certificate if one is sent. If the SSL or TLS client does not send a certificate, authentication fails only if the end of the channel acting as the SSL or TLS server is defined with either the SSLCAUTH parameter set to REQUIRED or an SSLPEER parameter value set. For more information, see Connecting two queue managers using SSL or TLS.

Working with SSL or TLS

These topics give instructions for performing single tasks related to using SSL or TLS with IBM MQ.

Many of them are used as steps in the higher-level tasks described in the following sections:
- "Identifying and authenticating users" on page 715
- "Authorizing access to objects" on page 735
- "Confidentiality of messages" on page 797
- "Data integrity of messages" on page 809
- "Keeping clusters secure" on page 813

Working with SSL or TLS on HP Integrity NonStop Server

Describes the IBM MQ client for HP Integrity NonStop Server OpenSSL security implementation, including security services, components, supported protocol versions, supported CipherSpecs, and unsupported security functionality.

IBM MQ SSL & TLS support provides the following security services for client channels:
- Authentication of the server and, optionally, authentication of the client.
- Encryption and decryption of the data that is flowing across a channel.
- Integrity checks on the data that is flowing across a channel.

The SSL and TLS support supplied with the IBM MQ client for HP Integrity NonStop Server comprises the following components:
- OpenSSL libraries and the **openssl** command.
- IBM MQ password stash command, **amqrsslc**.

The following required components for SSL or TLS client channel operation are not provided with the IBM MQ client for HP Integrity NonStop Server:
- An entropy daemon to provide a source of random data for OpenSSL cryptography.

Supported protocol versions

The IBM MQ client for HP Integrity NonStop Server supports the following protocol versions:
- SSL 3.0
- TLS 1.0
- TLS 1.2

Supported CipherSpecs

The IBM MQ client for HP Integrity NonStop Server supports the following CipherSpecs versions:
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- RC4_SHA_US
- RC4_MD5_US
- TRIPLE_DES_SHA_US
- TLS_RSA_WITH_3DES_EDE_CBC_SHA (deprecated)
- DES_SHA_EXPORT1024
- RC4_56_SHA_EXPORT1024
- RC4_MD5_EXPORT

- RC2_MD5_EXPORT
- DES_SHA_EXPORT
- TLS_RSA_WITH_DES_CBC_SHA
- NULL_SHA
- NULL_MD5
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_NULL_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- ECDHE_ECDSA_AES_128_CBC_SHA256
- ECDHE_ECDSA_AES_256_CBC_SHA384
- ECDHE_RSA_AES_128_CBC_SHA256
- ECDHE_RSA_AES_256_CBC_SHA384
- ECDHE_ECDSA_AES_128_GCM_SHA256
- ECDHE_ECDSA_AES_256_GCM_SHA384
- ECDHE_RSA_AES_128_GCM_SHA256
- ECDHE_RSA_AES_256_GCM_SHA384

Unsupported security functionality

The IBM MQ client for HP Integrity NonStop Server does not currently support:
- PKCS#11 Cryptographic hardware support
- LDAP Certificate Revocation List checking
- OCSP Online Certificate Status Protocol checking
- FIPS 140-2, NSA SUITE B cipher suite controls

Certificate management

Use a set of files to store digital certificate and certificate revocation information.

IBM MQ SSL and TLS support uses a set of files to store digital certificate and certificate revocation information. These files are located in a directory specified either programmatically by way of the KeyRepository field in the MQSCO structure passed on the MQCONNX call, by the *MQSSLKEYR* environment variable, or, in the SSL stanza of the `mqclient.ini` using the SSLKeyRepository attribute.

The MQSCO structure takes precedence over the MQSSLKEYR environment variable which takes precedence over the ini file stanza value.

Important: The key repository location specifies a directory location and not a filename on the HP Integrity NonStop Server platform.

The IBM MQ client for HP Integrity NonStop Server uses the following, case sensitive, named files in the key repository location:
- "Personal certificate store" on page 660
- "Certificate truststore" on page 660
- "Pass phrase stash file" on page 660
- "Certificate revocation list file" on page 661

Personal certificate store:

The personal certificate store file, `cert.pem`.

This file contains the personal certificate and the encrypted private key for the client to use, in PEM format. The existence of this file is optional when you are using SSL or TLS channels that do not require client authentication. Where client authentication is required by the channel, and SSLCAUTH(REQUIRED) is specified on the channel definition, this file must exist and contain both the certificate and encrypted private key.

File permissions must be set on this file to allow read access to the owner of the certificate store.

A correctly formatted `cert.pem` file must contain exactly two sections with the following headers and footers:

```
-----BEGIN PRIVATE KEY-----
Base 64 ASCII encoded private key data here
-----END PRIVATE KEY-----
```

```
-----BEGIN CERTIFICATE-----
Base 64 ASCII encoded certificate data here
-----END CERTIFICATE-----
```

The pass phrase for the encrypted private key is stored in the pass phrase stash file, `Stash.sth`.

Certificate truststore:

The certificate truststore file, `trust.pem`.

This file contains the certificates that are needed to validate the personal certificates that are used by queue managers that the client connects to, in PEM format. The certificate truststore is mandatory for all SSL or TLS client channels.

File permissions must be set to limit write access to this file.

A correctly formatted `trust.pem` file must contain one or more sections with the following headers and footers:

```
-----BEGIN CERTIFICATE-----
Base 64 ASCII encoded certificate data here
-----END CERTIFICATE-----
```

Pass phrase stash file:

The pass phrase stash file, `Stash.sth`.

This file is a binary format private to IBM MQ and contains the encrypted pass phrase for use when you are accessing the private key that is held in the `cert.pem` file. The private key itself is stored in the `cert.pem` certificate store.

This file is created or altered by using the IBM MQ **`amqrsslc`** command-line tool with the **`-s`** parameter. For example, where the directory `/home/alice` contains a `cert.pem` file:

```
amqrsslc -s /home/alice/cert

  Enter password for Keystore /home/alice/cert.pem :
    password


  Stashed the password in file /home/alice/Stash.sth
```

File permissions must be set on this file to allow read access to the owner of the associated personal certificate store.

Certificate revocation list file:

The certificate revocation list file, `crl.pem`.

This file contains the certificate revocation lists (CRLs) that the client uses to validate digital certificates, in PEM format. The existence of this file is optional. If this file is not present, no certificate revocation checks are done when you are validating certificates.

File permissions must be set to limit write access to this file.

A correctly formatted `crl.pem` file must contain one or more sections with the following headers and footers:

```
-----BEGIN X509 CRL-----
Base 64 ASCII encoded CRL data here
-----END X509 CRL-----
```

Working with SSL or TLS on IBM i

IBM i

This collection of topics gives instructions for individual tasks working with the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) in IBM MQ for IBM i.

For IBM i the SSL support is integral to the operating system. Ensure that you have installed the prerequisites listed in Hardware and software requirements on IBM i.

On IBM i, you manage keys and digital certificates with the Digital Certificate Manager (DCM) tool.

Accessing DCM

Follow these instructions to access the DCM interface.

About this task

Perform the following steps in a web browser that supports frames.

Procedure

1. Go to either `http://`*`machine`*`.domain:2001` or `https://`*`machine`*`.domain:2010`, where *machine* is the name of your computer.
2. Type a valid user profile and password when requested to. Ensure that your user profile has *ALLOBJ and *SECADM special authorities to enable you to create new certificate stores. If you do not have the special authorities, you can only manage your personal certificates or view the object signatures for the objects for which you are authorized. If you are authorized to use an object signing application, you can also sign objects from DCM.
3. On the Internet Configurations page, click **Digital Certificate Manager**. The Digital Certificate Manager page is displayed.

Assigning a certificate to a queue manager on IBM i

Use DCM to assign a certificate to a queue manager.

Use traditional IBM i digital certificate management to assign a certificate to a queue manager. This means that you can specify that a queue manager uses the system certificate store, and that the queue manager is registered for use as an application with Digital Certificate Manager. To do this, change the value of the queue manager **SSLKEYR** attribute to `*SYSTEM`.

When the **SSLKEYR** parameter is changed to `*SYSTEM`, IBM MQ registers the queue manager as a server application with a unique application label of QIBM_WEBSPHERE_MQ_QMGRNAME and a label with a description of Qmgrname (WMQ). Note that channel **CERTLABL** attributes are not used if you use the *SYSTEM certificate store. The queue manager then appears as a server application in Digital Certificate Manager, and you can assign to this application any server or client certificate in the system store.

Because the queue manager is registered as an application, advanced features of DCM such as defining CA trust lists can be carried out.

If the **SSLKEYR** parameter is changed to a value other than `*SYSTEM`, IBM MQ deregisters the queue manager as an application with Digital Certificate Manager. If a queue manager is deleted, it is also deregistered from DCM. A user with sufficient *SECADM authority can also manually add or remove applications from DCM.

Setting up a key repository on IBM i

A key repository must be set up at both ends of the connection. The default certificate stores can be used or you can create your own.

An SSL or TLS connection requires a *key repository* at each end of the connection. Each queue manager and IBM MQ MQI client must have access to a key repository. If you want to access the key repository using a file name and password (that is, not using the *SYSTEM option) ensure that the QMQM user profile has the following authorities:

- Execute authority for the directory containing the key repository
- Read authority for the file containing the key repository

See "The SSL or TLS key repository" on page 398 for more information. Note that channel **CERTLABL** attributes are not used if you use the *SYSTEM certificate store.

On IBM i, digital certificates are stored in a certificate store that is managed with DCM. These digital certificates have labels, which associate a certificate with a queue manager or an IBM MQ MQI client. SSL and TLS use the certificates for authentication purposes.

The label is either the value of the **CERTLABL** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client user logon ID appended, all in lowercase. See Digital certificate labels for details.

The queue manager or IBM MQ MQI client certificate store name comprises a path and stem name. The default path is `/QIBM/UserData/ICSS/Cert/Server/` and the default stem name is `Default`. On IBM i, the default certificate store, `/QIBM/UserData/ICSS/Cert/Server/Default.kdb`, is also known as `*SYSTEM`. Optionally, you can define your own path and stem name.

If you define your own path or file name, set the permissions to the file to tightly control access to it.

"Changing the key repository location for a queue manager on IBM i" on page 664 tells you about specifying the certificate store name. You can specify the certificate store name either before or after creating the certificate store.

Note: The operations you can perform with DCM might be limited by the authority of your user profile. For example, you require *ALLOBJ and *SECADM authorities to create a CA certificate.

Creating a certificate store on IBM i:

If you do not want to use the default certificate store, follow this procedure to create your own.

About this task

Create a new certificate store only if you do not want to use the IBM i default certificate store.

To specify that the IBM i system certificate store is to be used, change the value of the queue manager's SSLKEYR attribute to *SYSTEM. This value indicates that the queue manager uses the system certificate store, and the queue manager is registered for use as an application with Digital Certificate Manager (DCM).

Procedure
1. Access the DCM interface, as described in "Accessing DCM" on page 661
2. In the navigation panel, click **Create New Certificate Store**. The Create New Certificate Store page is displayed in the task frame.
3. In the task frame, select **Other System Certificate Store** and click **Continue**. The Create a Certificate in New Certificate Store page is displayed in the task frame.
4. Select **No - Do not create a certificate in the certificate store** and click **Continue**. The Certificate Store Name and Password page is displayed in the task frame.
5. In the **Certificate store path and filename** field, type an IFS path and file name, for example `/QIBM/UserData/mqm/qmgrs/qm1/key.kdb`
6. Type a password in the **Password** field and type it again in the **Confirm Password** field. Click **Continue**. Make a note of the password (which is case sensitive) because you need it when you stash the repository key.
7. To exit from DCM, close your browser window.

What to do next

When you have created the certificate store using DCM, ensure you stash the password, as described in "Stashing the certificate store password on IBM i systems"
Related tasks:
"Importing a certificate into a key repository on IBM i" on page 669
Follow this procedure to import a certificate.

Stashing the certificate store password on IBM i systems:

Stash the certificate store password by using CL commands.

The following instructions apply to stashing the certificate store password on IBM i for a queue manager. Alternatively, for an IBM MQ MQI client, if you are not using the *SYSTEM certificate store (that is, the MQSSLKEYR environment is set to a value other than *SYSTEM), follow the procedure described in the "Stash the certificate store password" on page 672 section of "IBM MQ SSL Client utility (amqrsslc) for IBM i" on page 671.

If you have specified that the *SYSTEM certificate store is to be used (by changing the value of the SSLKEYR attribute of the queue manager to *SYSTEM) you must not follow these steps.

When you have created the certificate store using DCM, use the following commands to stash the password:

```
STRMQM MQMNAME('queue manager name')
CHGMQM MQMNAME('queue manager name') SSLKEYRPWD('password')
```

The password, which is case sensitive, must be entered in single quotation marks exactly as you entered it in step 6 of "Creating a certificate store on IBM i" on page 663.

Note: If you are not using the default system certificate store, and you do not stash the password, attempts to start SSL channels fail because they cannot obtain the password required to access the certificate store.

Locating the key repository for a queue manager on IBM i

Use this procedure to obtain the location of your queue manager's certificate store.

Procedure

1. Display your queue manager's attributes, using the following command:
   ```
   DSPMQM MQMNAME('queue manager name')
   ```
2. Examine the command output for the path and stem name of the certificate store. For example: `/QIBM/UserData/ICSS/Cert/Server/Default`, where `/QIBM/UserData/ICSS/Cert/Server` is the path and `Default` is the stem name.

Changing the key repository location for a queue manager on IBM i

Change the location of your queue manager's certificate store using either CHGMQM or ALTER QMGR.

Procedure

Use either the CHGMQM command or the ALTER QMGR MQSC command to set your queue manager's key repository attribute.

1. Using CHGMQM: `CHGMQM MQMNAME('qm1') SSLKEYR('/QIBM/UserData/ICSS/Cert/Server/MyKey')`
2. Using ALTER QMGR: `ALTER QMGR SSLKEYR('/QIBM/UserData/ICSS/Cert/Server/MyKey')`

In either case, the certificate store has the fully qualified file name: `/QIBM/UserData/ICSS/Cert/Server/MyKey.kdb`

What to do next

When you change the location of a queue manager's certificate store, certificates are not transferred from the old location. If the CA certificates preinstalled when you create the certificate store are insufficient, you must populate the new certificate store with certificates, as described in "Importing a certificate into a key repository on IBM i" on page 669. You must also stash the password for the new location, as described in "Stashing the certificate store password on IBM i systems" on page 663.

Creating a certificate authority and certificate for testing on IBM i

Use this procedure to create a local CA certificate to sign certificate requests, and to create and install the CA certificate.

Before you begin

The instructions in this topic assume that a local certificate authority (CA) does not exist. If a local CA does exist, go to "Requesting a server certificate on IBM i" on page 666.

About this task

The CA certificates that are provided when you install SSL are signed by the issuing CA. On IBM i, you can generate a local certificate authority that can sign server certificates for testing SSL communications on your system. Follow these steps in a Web browser to create a local CA certificate:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 661.
2. In the navigation panel, click **Create a Certificate Authority**. The Create a Certificate Authority page is displayed in the task frame.
3. Type a password in the **Certificate store password** field and type it again in the **Confirm password** field.
4. Type a name in the **Certificate Authority (CA) name** field, for example `SSL Test Certificate Authority`.
5. Type appropriate values in the **Common Name** and **Organization** fields, and select a country. For the remaining optional fields, type the values you require.
6. Type a validity period for the local CA in the **Validity period** field. The default value is 1095 days.
7. Click **Continue**. The CA is created, and DCM creates a certificate store and a CA certificate for your local CA.
8. Click **Install certificate**. The download manager dialog box is displayed.
9. Type the full path name for the temporary file in which you want to store the CA certificate and click **Save**.
10. When download is complete, click **Open**. The Certificate window is displayed.
11. Click **Install certificate**. The Certificate Import wizard is displayed.
12. Click **Next**.
13. Select **Automatically select the certificate store based on the type of certificate** and click **Next**.
14. Click **Finish**. A confirmation window is displayed.
15. Click **OK**.
16. In the Certificate window, click **OK**.
17. Click **Continue**. The Certificate Authority Policy page is displayed in the task frame.
18. In the **Allow creation of user certificates** field, select **Yes**.
19. In the **Validity period** field, type the validity period of certificates that are issued by your local CA. The default value is 365 days.
20. Click **Continue**. The Create a Certificate in New Certificate Store page is displayed in the task frame.
21. Check that none of the applications are selected.
22. Click **Continue** to complete the setup of the local CA.

Requesting a server certificate on IBM i

Digital certificates protect against impersonation, certifying that a public key belongs to a specified entity. A new server certificate can be requested from a certificate authority using the Digital Certificate Manager (DCM).

About this task

Perform the following steps in a Web browser:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 661.
2. In the navigation panel, click **Select a Certificate Store**. The Select a Certificate Store page is displayed in the task frame.
3. Select the certificate store you want to use and click **Continue**.
4. Optional: If you selected ***SYSTEM** in step 3, enter the system store password and click **Continue**.
5. Optional: If you selected **Other System Certificate Store** in step 3, in the **Certificate store path and filename** field, type the IFS path and file name you set when you created your certificate store. Also type a password in the **Certificate Store Password** field. Then click **Continue**
6. In the navigation panel, click **Create Certificate**.
7. In the task frame, select the **Server or client certificate** radio button and click **Continue**. The Select a Certificate Authority (CA) page is displayed in the task frame.
8. If you have a local CA on your workstation you choose either the local CA or a commercial CA to sign the certificate. Select the radio button for the CA you want and click **Continue**. The Create a Certificate page is displayed in the task frame.
9. Optional: For a queue manager, in the **Certificate label** field, enter the certificate label. The label is either the value of the **`CERTLABL`** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager appended, all in lowercase. See Digital certificate labels for details. For example, for queue manager QM1, type `ibmwebspheremqqm1` to use the default value.
10. Optional: For an IBM MQ MQI client, in the **Certificate label** field, type `ibmwebspheremq` followed by your logon user ID folded to lowercase. For example, type `ibmwebspheremqmyuserID`
11. Type appropriate values in the **Common Name** and **Organization** fields, and select a country. For the remaining optional fields, type the values you require.

Results

If you selected a commercial CA to sign your certificate, DCM creates a certificate request in PEM (Privacy-Enhanced Mail) format. Forward the request to your chosen CA.

If you selected the local CA to sign your certificate, DCM informs you that the certificate has been created in the certificate store and can be used.

Requesting a server certificate on IBM i for IBM Key Manager

Follow this procedure to create a certificate signed by your local certificate authority (CA), or to apply for a server certificate signed by a commercial CA for import into the IBM Key Management (iKeyman) utility.

About this task

A user certificate must be used when the Digital Certificate Manager (DCM) serves as the certificate manager for IBM MQ on multiple platforms. For personal certificates distributed to other platforms and for import into the iKeyman utility, perform the following steps in a Web browser:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 661.
2. In the navigation pane, click **Create Certificate**. The Create Certificate page is displayed in the task frame.
3. On the Create Certificate panel, select the **User certificate** radio button and click **Continue**. The Create User Certificate page is displayed.
4. On the Create User Certificate panel, complete the required fields under Certificate Information for **Organization name**, **State** or **province**, **Country** or **region**. Optionally, put values in the **Organization unit** and **Locality** or **city** fields. Click **Continue**. The **Common name** is automatically set to the user ID with which you are logged on to the iSeries system.
5. On the next Create User Certificate panel, click **Install certificate** and click **Continue**. A message is displayed stating, `Your personal certificate has been installed. You should keep a backup copy of this certificate.`
6. Click **OK**.
7. Depending on the internet browser you used to access DCM, do the following steps:
   a. For Internet Explorer choose: **Tools>Internet Options>Content tab>Certificates button>Personal tab>**. Select the certificate and click **Export**.
   b. For Mozilla Firefox choose: **Tools>Options>Advanced>Encryption tab>View Certificates button>Your Certificates tab>**. Select the certificate and click **Backup**. Select the path and filename and click **OK**.
8. Transfer the exported certificate to the remote system using FTP in binary format.
9. Add the exported certificate from step 7 to the iKeyman utility in the key database.
   a. If the certificate was saved using Internet Explorer, use the instructions described in Importing from a Microsoft `.pfx` file.
   b. If the certificate was saved using Mozilla Firefox, use the instructions described in Importing a personal certificate into a key repository.

   During the import, ensure that the label name of the personal certificate and the signer certificate are changed to what IBM MQ is expecting. The label must be either the value of the IBM MQ **`CERTLABL`** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager appended, all in lowercase. See Digital certificate labels for details.

Adding server certificates to a key repository on IBM i

Follow this procedure to add a requested certificate to the key repository.

About this task

After the CA sends you a new server certificate, you add it to the certificate store from which you generated the request. If the CA sends the certificate as part of an email message, copy the certificate into a separate file.

Note:

- You do not need to perform this procedure if the server certificate is signed by your local CA.
- Before you import a server certificate in PKCS #12 format into DCM, you must first import the corresponding CA certificate.

Use the following procedure to receive a server certificate into the queue manager certificate store:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 661.
2. In the **Manage Certificates** task category in the navigation panel, click **Import Certificate**. The Import Certificate page displays in the task frame.
3. Select the radio button for your certificate type and click **Continue**. Either the Import Server or Client Certificate page or the Import Certificate Authority (CA) Certificate page displays in the task frame.
4. In the **Import File** field, type the file name of the certificate you want to import and click **Continue**. DCM automatically determines the format of the file.
5. If the certificate is a **Server or client** certificate, type the password in the task frame and click **Continue**. DCM informs you that the certificate has been imported.

Exporting a certificate from a key repository on IBM i

Exporting a certificate exports both the public and private key. This action should be taken with extreme caution, since passing on a private key would completely compromise your security.

Before you begin

When you share a user's certificate with another user, you exchange public keys. This process is described in **Task 5. Sharing Certificates** in the Quick Start Guide for IBM MQ AMS on UNIX platforms . When you export a certificate as described here, you export both the public and private key. This action should be taken with extreme caution, since passing on a private key would completely compromise your security.

About this task

Perform the following steps on the computer from which you want to export the certificate:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 661.
2. In the navigation panel, click **Select a Certificate Store**. The Select a Certificate Store page is displayed in the task frame.
3. Select the certificate store you want to use and click **Continue**.
4. Optional: If you selected ***SYSTEM** in step 3, enter the system store password and click **Continue**.
5. Optional: If you selected **Other System Certificate Store** in step 3, in the **Certificate store path and filename** field, type the IFS path and file name you set when you created your certificate store and type a password in the **Certificate Store Password** field. Then click **Continue**

6. In the **Manage Certificates** task category in the navigation panel, click **Export Certificate**. The Export a Certificate page is displayed in the task frame.
7. Select the radio button for your certificate type and click **Continue**. Either the Export Server or Client Certificate page or the Export Certificate Authority (CA) Certificate page is displayed in the task frame.
8. Select the certificate you want to export.
9. Select the radio button to specify whether you want to export the certificate to a file or directly into another certificate store.
10. If you selected to export a server or client certificate to a file, provide the following information:
    - The path and file name of the location where you want to store the exported certificate.
    - For a personal certificate, the password that is used to encrypt the exported certificate and the target release. For CA certificates, you do not need to specify the password.
11. If you selected to export a certificate directly into another certificate store, specify the target certificate store and its password.
12. Click **Continue**.

Importing a certificate into a key repository on IBM i

Follow this procedure to import a certificate.

Before you begin

Before you import a personal certificate in PKCS #12 format into DCM, you must first import the corresponding CA certificate.

About this task

Perform these steps on the machine to which you want to import the certificate.

Procedure

1. Access the DCM interface, as described in “Accessing DCM” on page 661.
2. In the navigation panel, click **Select a Certificate Store**. The Select a Certificate Store page is displayed in the task frame.
3. Select the certificate store you want to use and click **Continue**.
4. Optional: If you selected ***SYSTEM** in step 3, enter the system store password and click **Continue**.
5. Optional: If you selected **Other System Certificate Store** in step 3, in the **Certificate store path and filename** field, type the IFS path and file name you set when you created your certificate store and type a password in the **Certificate Store Password** field. Then click **Continue**
6. In the **Manage Certificates** task category in the navigation panel, click **Import Certificate**. The Import Certificate page is displayed in the task frame.
7. Select the radio button for your certificate type and click **Continue**. Either the Import Server or Client Certificate page or the Import Certificate Authority (CA) Certificate page is displayed in the task frame.
8. In the **Import File** field, type the file name of the certificate you want to import and click **Continue**. DCM automatically determines the format of the file.
9. If the certificate is a **Server or client** certificate, type the password in the task frame and click **Continue**. DCM informs you that the certificate has been imported.

Removing certificates in IBM i

Use this procedure to remove personal certificates.

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 661.
2. In the navigation panel, click **Select a Certificate Store**. The Select a Certificate Store page is displayed in the task frame.
3. Select the **Other System Certificate Store** check box and click **Continue**. The Certificate Store and Password page is displayed.
4. In the **Certificate store path and filename** field, type the IFS path and file name you set when you created the certificate store.
5. Type a password in the **Certificate Store Password** field. Click **Continue**. The Current Certificate Store page is displayed in the task frame.
6. In the **Manage Certificates** task category in the navigation panel, click **Delete Certificate**. The Confirm Delete Certificate page is displayed in the task frame.
7. Select the certificate you want to delete. Click **Delete**.
8. Click **Yes** to confirm that you want to delete the certificate. Otherwise, click **No**. DCM informs you if it has deleted the certificate.

Using the *SYSTEM certificate store for one-way authentication on IBM i

Follow these instructions to set up one-way authentication.

Before you begin

- Create a queue manager, channels, and transmission queues.
- Create a server or client certificate on the server queue manager.
- Transfer the CA certificate to the client queue manager and imported it into the key repository.
- Start a listener on the server and client queue managers.

About this task

To use one-way authentication, using a computer running IBM i as the SSL server, set the SSL Key Repository (SSLKEYR) parameter to *SYSTEM. This setting registers the IBM MQ queue manager as an application. You can then assign a certificate to the queue manager to enable one-way authentication.

You can also use private keystores to implement one-way authentication by creating a dummy certificate for the client queue manager in the key repository.

Procedure

1. Perform the following steps on the server and client queue managers:
   a. Alter the queue manager to set the SSLKEYR parameter by issuing the command `CHGMQM MQMNAME(SSL) SSLKEYR(*SYSTEM)`.
   b. Stash the password for the default key repository by issuing the command `CHGMQM MQMNAME(SSL) SSLKEYRPWD('xxxxxxx')`. The password must be in single quotation marks.
   c. Alter the channels to have the correct CipherSpec in the SSLCIPHER parameter.
   d. Refresh SSL security by issuing the command `RFRMQMAUT QMNAME(QMGRNAME) TYPE(*SSL)`.
2. Assign the certificate to the server queue manager using DCM, as follows:
   a. Access the DCM interface, as described in "Accessing DCM" on page 661.
   b. In the navigation panel, click **Select a Certificate Store**. The Select a Certificate Store page is displayed in the task frame.
   c. Select the *SYSTEM certificate store and click **Continue**.

d. In the left panel, expand **Manage Applications**.

e. Select the **View Application** definition to check that the queue manager has been registered as an application. `SSL (WMQ)` is listed in the table.

f. Select **Update Certificate Assignment**.

g. Select **Server** and click **Continue**.

h. Select QMGRNAME (WMQ) and click **Update certificate assignment**.

i. Select the certificate and click **Assign New Certificate**. A window opens stating that the certificate has been assigned to the application.

IBM MQ SSL Client utility (amqrsslc) for IBM i

The IBM MQ SSL Client utility (amqrsslc) for IBM i is used by the IBM MQ MQI client on IBM i systems to register or unregister the client user profile, or stash the certificate store password. The utility can only be run by a user with a profile with *ALLOBJ special authority or a member of QMQMADM that has options to create or delete application registrations in the Digital Certificate Manager (DCM).

Syntax diagram

```
►►—amqrsslc——┬—-s—PathOfKeyDatabase—————————┬——————————————————————————————————►◄
             │     ┌—current user—┐          │
             ├—-r——┤              ├——————————┤
             │     └—UserProfile——┘          │
             │     ┌—current user—┐          │
             └—-u——┤              ├——————————┘
                   └—UserProfile——┘
```

Register the client user profile

If the IBM MQ MQI client is using the *SYSTEM certificate store, you must register the client user profile (logon user) for use as an application with Digital Certificate Manager (DCM).

If you want to register the client user profile, run the **amqrsslc** program with the `-r` option with *UserProfile*. The user profile used when calling **amqrsslc** must have *USE authority. Providing *UserProfile* with the `-r` option registers the *UserProfile* as a server application with a unique application label of QIBM_WEBSPHERE_MQ_< *UserProfile* > and a label with a description of < *UserProfile* > (WMQ). This server application then is displayed in the DCM, and you can assign to this application any server or client certificate in the system store.

Note: If a user profile is not specified with `-r` option, then the user profile of the user running the **amqrsslc** tool is registered.
The following code uses **amqrsslc** to register a user profile. In the first example, the specified user profile is registered; in the second it is the profile of the logged in user:

```
CALL PGM(QMQM/AMQRSSLC) PARM('-r' UserProfile)
CALL PGM(QMQM/AMQRSSLC) PARM('-r')
```

Unregister the client user profile

To unregister the client profile, run the **amqrsslc** program with the `-u` option with *UserProfile*. The user profile used when calling **amqrsslc** must have *USE authority. Providing the *UserProfile* with the `-u` option unregisters *UserProfile* with label QIBM_WEBSPHERE_MQ_< *UserProfile* > from the DCM.

Note: If a user profile is not specified with `-u` option, then the user profile of the user running the **amqrsslc** tool is unregistered.

The following code uses **amqrsslc** to unregister a user profile. In the first example, the specified user profile is unregistered; in the second it is the profile of the logged in user:

```
CALL PGM(QMQM/AMQRSSLC) PARM('-u' UserProfile)
CALL PGM(QMQM/AMQRSSLC) PARM('-u')
```

Stash the certificate store password

If the IBM MQ MQI client is not using the *SYSTEM certificate store and using another certificate store (that is, MQSSLKEYR is set to value other than *SYSTEM), then the password of the key database must be stashed. Use `-s` option for stashing the password of key database.

In the following code, the fully qualified file name of the certificate store is `/Path/Of/KeyDatabase/MyKey.kdb`:

```
CALL PGM(QMQM/AMQRSSLC) PARM('-s' '/Path/Of/KeyDatabase/MyKey')
```

Running this code results in a request for the password of this key database. This password is stashed in a file with the same name as key database with a .sth extension. This file is stored on the same path as the key database. The code example generates a stash file of `/Path/Of/KeyDatabase/MyKey.sth`. QMQM is the user owner and QMQMADM the group owner for this file. QMQM and QMQMADM have read, write permission, and other profiles have only read permission.

When changes to certificates or the certificate store become effective on IBM i

When you change the certificates in a certificate store, or the location of the certificate store, the changes take effect depending on the type of channel and how the channel is running.

Changes to the certificates in the certificate store and to the key repository attribute become effective in the following situations:

- When a new outbound single channel process first runs an SSL channel.
- When a new inbound TCP/IP single channel process first receives a request to start an SSL channel.
- When the MQSC command REFRESH SECURITY TYPE(SSL) is issued to refresh the IBM MQ SSL environment.
- For client application processes, when the last SSL connection in the process is closed. The next SSL connection picks up the certificate changes.
- For channels that run as threads of a process pooling process (amqrmppa), when the process pooling process is started or restarted and first runs an SSL channel. If the process pooling process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).
- For channels that run as threads of the channel initiator, when the channel initiator is started or restarted and first runs an SSL channel. If the channel initiator process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).
- For channels that run as threads of a TCP/IP listener, when the listener is started or restarted and first receives a request to start an SSL channel. If the listener has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).

Configuring cryptographic hardware on IBM i

Use this procedure to configure the 4758 PCI Cryptographic Coprocessor on IBM i

Before you begin

Ensure your user profile has *ALLOBJ and *SECADM special authorities to enable you to configure the coprocessor hardware.

Procedure

1. Go to either `http://machine.domain:2001` or `https://machine.domain:2010`, where *machine* is the name of your computer. A dialog box is displayed, requesting a user name and a password.
2. Type a valid IBM i user profile and password.
3. On the AS/400 Tasks page, click **4758 PCI Cryptographic Coprocessor**.

What to do next

For more information about configuring the 4758 PCI Cryptographic Coprocessor, refer to the *IBM i product documentation* at IBM System i and IBM i product documentation.

Working with SSL or TLS on UNIX, Linux and Windows systems

On UNIX, Linux and Windows systems, Secure Sockets Layer (SSL) or Transport Layer Security (TLS) support is installed with IBM MQ.

For more detailed information about certificate validation policies, see Certificate validation and trust policy design.

Using runmqckm, runmqakm, and strmqikm to manage digital certificates

On UNIX, Linux and Windows systems, manage keys and digital certificates with the **strmqikm** (iKeyman) GUI, or from the command line using **runmqckm** (iKeycmd) or **runmqakm** (GSKCapiCmd).

- For **UNIX and Linux** systems:
  - Use the **strmqikm** (iKeyman) command to start the iKeyman GUI.
  - Use the **runmqckm** (iKeycmd) command to perform tasks with the iKeycmd command line interface.
  - Use the **runmqakm** (GSKCapiCmd) command to perform tasks with the runmqakm command line interface. The command syntax for **runmqakm** is the same as the syntax for **runmqckm**.

    If you need to manage SSL certificates in a way that is FIPS compliant, use the **runmqakm** command instead of the **runmqckm** or **strmqikm** commands.

  See Managing keys and certificates for a full description of the command line interfaces for the **runmqckm** and **runmqakm** commands.

  If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

  See GSKit: PKCS#11 and JRE addressing mode for further information.

  Before you run the **strmqikm** command to start the iKeyman GUI, ensure you are working on a machine that is able to run the X Window System and that you do the following:
  - Set the DISPLAY environment variable, for example:

    ```
    export DISPLAY=mypc:0
    ```
  - Ensure that your PATH environment variable contains **/usr/bin** and **/bin**. This is also required for the **runmqckm** and **runmqakm** commands. For example:

    ```
    export PATH=$PATH:/usr/bin:/bin
    ```

- For **Windows** systems:
  - Use the **`strmqikm`** command to start the iKeyman GUI.
  - Use the **`runmqckm`** command to perform tasks with the iKeycmd command line interface.

    If you need to manage SSL certificates in a way that is FIPS compliant, use the **`runmqakm`** command instead of the **`runmqckm`** or **`strmqikm`** commands.

To request SSL tracing on UNIX, Linux or Windows systems, see strmqtrc.

Related information:

runmqckm, and runmqakm commands

Setting up a key repository on UNIX, Linux, and Windows systems

You can set up a key repository by the using **`strmqikm`** (iKeyman) GUI, or from the command line using **`runmqckm`** (iKeycmd) or **`runmqakm`** (GSKCapiCmd) commands.

About this task

An SSL or TLS connection requires a *key repository* at each end of the connection. Each IBM MQ queue manager and IBM MQ MQI client must have access to a key repository. For more information, see "The SSL or TLS key repository" on page 398.

On UNIX, Linux, and Windows systems, digital certificates are stored in a key database file that is managed by using the **`iKeyman`** user interface, or by using the **`iKeycmd`** or **`runmqakm`** commands. These digital certificates have labels. A specific label associates a personal certificate with a queue manager or IBM MQ MQI client. SSL and TLS use that certificate for authentication purposes. On UNIX, Linux, and Windows systems, IBM MQ uses either the value of the **`CERTLABL`** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client user logon ID appended, all in lowercase. See Digital certificate labels for details.

The key database file name comprises a path and stem name:

- On UNIX and Linux systems, the default path for a queue manager (set when you created the queue manager) is `/var/mqm/qmgrs/<queue_manager_name>/ssl`.

  On Windows systems, the default path is `*MQ_INSTALLATION_PATH*\Qmgrs\*queue_manager_name*\ssl`, where `*MQ_INSTALLATION_PATH*` is the directory in which IBM MQ is installed. For example, `C:\Program Files\IBM\WebSphere MQ\Qmgrs\QM1\ssl`.

  The default stem name is `key`. Optionally, you can choose your own path and stem name, but the extension must be `.kdb`.

  If you choose your own path or file name, set the permissions to the file to tightly control access to it.
- For an IBM MQ client, there is no default path or stem name. Tightly control access to this file. The extension must be `.kdb`.

Do not create key repositories on a file system that does not support file level locks, for example NFS version 2 on Linux systems.

See "Changing the key repository location for a queue manager on UNIX, Linux or Windows systems" on page 679 for information about checking and specifying the key database file name. You can specify the key database file name either before or after creating the key database file.

The user ID from which you run the **`iKeyman`** or **`iKeycmd`** commands must have write permission for the directory in which the key database file is created or updated. For a queue manager using the default `ssl` directory, the user ID from which you run **`iKeyman`** or **`iKeycmd`** must be a member of the mqm group. For an IBM MQ MQI client, if you run **`iKeyman`** or **`iKeycmd`** from a user ID different from that under which the client runs, you must alter the file permissions to enable the IBM MQ MQI client to access the key database file at run time. For more information, see "Accessing and securing your key database files on Windows" on page 676 or "Accessing and securing your key database files on UNIX and Linux systems" on page 677.

In **iKeyman** or **iKeycmd** Version 7.0, new key databases are automatically populated with a set of pre-defined certificate authority (CA) certificates. In **iKeyman** or **iKeycmd** Version 8.0, key databases are not automatically populated, making the initial setup more secure because you include only the CA certificates that you want, in your key database file.

Note: Because of this change in behavior for GSKit Version 8.0 that results in CA certificates no longer being automatically added to the repository, you must manually add your preferred CA certificates. This change of behavior provides you with more granular control over the CA certificates used. See "Adding default CA certificates into an empty key repository, on UNIX, Linux or Windows systems with GSKit Version 8.0" on page 677.

Procedure

Note: If you must manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command. The **iKeyman** user interface does not provide a FIPS-compliant option.

- To create a key database by using the iKeyman user interface, complete the following steps:
  1. On UNIX and Linux systems, log in as the root user. On Windows systems, log in as Administrator or as a member of the MQM group.
  2. Start the iKeyman user interface by running the **strmqikm** command.
  3. From the **Key Database File** menu, click **New**. The New window opens.
  4. Click **Key database type** and select **CMS** (Certificate Management System).
  5. In the **File Name** field, type a file name. This field already contains the text `key.kdb`. If your stem name is `key`, leave this field unchanged. If you specified a different stem name, replace `key` with your stem name. However, you must not change the `.kdb` extension.
  6. In the **Location** field, type the path. For example:
     - For a queue manager: `/var/mqm/qmgrs/QM1/ssl` (on UNIX and Linux systems) or `C:\ProgramData\IBM\MQ\qmgrs\QM1\ssl` (on Windows systems).

       The path must match the value of the **SSLKeyRepository** attribute of the queue manager.
     - For an IBM MQ client: `/var/mqm/ssl` (on UNIX and Linux systems) or `C:\mqm\ssl` (on Windows systems).
  7. Click **OK**. The Password Prompt window opens.
  8. Type a password in the **Password** field, and type it again in the **Confirm Password** field.
  9. Select the **Stash the password to a file** check box.

     Note: If you do not stash the password, attempts to start SSL or TLS channels fail because they cannot obtain the password required to access the key database file.
  10. Click **OK**. The Personal Certificates window opens.
  11. Set the access permissions as described in "Accessing and securing your key database files on Windows" on page 676 or "Accessing and securing your key database files on UNIX and Linux systems" on page 677.
- To create a key database by using the command line, use either of the following commands:
  - On UNIX, Linux, and Windows systems:
    ```
    runmqckm -keydb -create -db filename -pw password -type cms -stash
    ```
  - Using runmqakm:
    ```
    runmqakm -keydb -create -db filename -pw password -type cms
    -stash -fips -strong
    ```

  where:

-db ***filename***
: Specifies the fully qualified file name of a CMS key database, and must have a file extension of `.kdb`.

-pw ***password***
: Specifies the password for the CMS key database.

-type ***cms***
: Specifies the type of database. (For IBM MQ, it must be `cms`.)

-stash
: Saves the key database password to a file.

-fips
: Specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-strong
: Checks that the password entered satisfies the minimum requirements for password strength. The minimum requirements for a password are as follows:
  - The password must be a minimum length of 14 characters.
  - The password must contain a minimum of one lowercase character, one uppercase character, and one digit or special character. Special characters include the asterisk (*), the dollar sign ($), the number sign (#), and the percent sign (%). A space is classified as a special character.
  - Each character can occur a maximum of three times in a password.
  - A maximum of two consecutive characters in the password can be identical.
  - All characters are in the standard ASCII printable character set, within the range 0x20 - 0x7E.

Accessing and securing your key database files on Windows:

The key database files might not have appropriate access permissions. You must set appropriate access to these files.

Set access control to the files *key*.kdb, *key*.sth, *key*.crl, and *key*.rdb, where *key* is the stem name of your key database, to grant authority to a restricted set of users.

Consider granting access as follows:

full authority
: BUILTIN\Administrators, NT AUTHORITY\SYSTEM, and the user who created the database files.

read authority
: For a queue manager, the local mqm group only. This assumes that the MCA is running under a user ID in the mqm group.

  For a client, the user ID under which the client process is running.

Accessing and securing your key database files on UNIX and Linux systems:

The key database files might not have appropriate access permissions. You must set appropriate access to these files.

For a queue manager, set permissions on the key database files so that queue manager and channel processes can read them when necessary, but other users cannot read or modify them. Normally, the mqm user needs read permissions. If you have created the key database file by logging in as the mqm user, then the permissions are probably sufficient; if you were not the mqm user, but another user in the mqm group, you probably need to grant read permissions to other users in the mqm group.

Similarly for a client, set permissions on the key database files so that client application processes can read them when necessary, but other users cannot read or modify them. Normally, the user under which the client process runs needs read permissions. If you have created the key database file by logging in as that user, then the permissions are probably sufficient; if you were not the client process user, but another user in that group, you probably need to grant read permissions to other users in the group.

Set the permissions on the files `*key*.kdb`, `*key*.sth`, `*key*.crl`, and `*key*.rdb`, where *key* is the stem name of your key database, to read and write for the file owner, and to read for the mqm or client user group (-rw-r-----).

Adding default CA certificates into an empty key repository, on UNIX, Linux or Windows systems with GSKit Version 8.0:

Follow this procedure to add one or more of the default CA certificates to an empty key repository with GSKit version 8.

In GSKit Version 7.0, the behavior when creating a new key repository was to automatically add in a set of default CA certificates for commonly-used Certificate Authorities. For GSKit version 8, this behavior has changed so that CA certificates are no longer automatically added to the repository. The user is now required to manually add CA certificates into the key repository.

Using iKeyman

Perform the following steps on the machine on which you want to add the CA certificate:
1. Start the iKeyman GUI using the **`strmqikm`** command (on UNIX, Linux and Windows systems).
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file to which you want to add the certificate, for example `key.kdb`.
6. Click **Open**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file displays in the **File Name** field.
8. In the **Key database content** field, select **Signer Certificates**.
9. Click **Populate**. The Add CA's Certificate window opens.
10. The CA certificates that are available to be added to the repository are displayed in a hierarchical tree structure. Select the top level entry for the organization whose CA certificates you wish to trust to view the complete list of valid CA certificates.
11. Select the CA certificates you wish to trust from the list and click **OK**. The certificates are added to the key repository.

Using the command line

Use the following commands to list, then add CA certificates using iKeycmd:

- Issue the following command to list the default CA certificates along with the organizations which issue them:

  ```
  runmqckm -cert -listsigners
  ```

- Issue the following command to add all of the CA certificates for the organization specified in the *label* field:

  ```
  runmqckm -cert -populate -db filename -pw password -label label
  ```

where:

`-db` *filename*	is the fully qualified path name of the key database.
`-pw` *password*	is the password for the key database.
`-label` *label*	is the label attached to the certificate.

Note: Adding a CA certificate to a key repository results in IBM MQ trusting all personal certificates signed by that CA certificate. Consider carefully which Certificate Authorities you wish to trust and only add the set of CA certificates needed to authenticate your clients and managers. It is not recommended to add the full set of default CA certificates unless this is a definitive requirement for your security policy.

Locating the key repository for a queue manager on UNIX, Linux or Windows systems

Use this procedure to obtain the location of your queue manager's key database file

Procedure

1. Display your queue manager's attributes, using either of the following MQSC commands:

   ```
   DISPLAY QMGR ALL
   DISPLAY QMGR SSLKEYR
   ```

   You can also display your queue manager's attributes using the IBM MQ Explorer or PCF commands.
2. Examine the command output for the path and stem name of the key database file. For example,
   a. on UNIX and Linux systems: `/var/mqm/qmgrs/QM1/ssl/key`, where `/var/mqm/qmgrs/QM1/ssl` is the path and `key` is the stem name
   b. on Windows: *MQ_INSTALLATION_PATH*`\qmgrs\QM1\ssl\key`, where *MQ_INSTALLATION_PATH*`\qmgrs\QM1\ssl` is the path and `key` is the stem name. *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.

Changing the key repository location for a queue manager on UNIX, Linux or Windows systems

You can change the location of your queue manager's key database file by various means including the MQSC command ALTER QMGR.

You can change the location of your queue manager's key database file by using the MQSC command ALTER QMGR to set your queue manager's key repository attribute. For example, on UNIX and Linux systems:

```
ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QM1/ssl/MyKey')
```

The key database file has the fully qualified file name: `/var/mqm/qmgrs/QM1/ssl/MyKey.kdb`

On Windows:

```
ALTER QMGR SSLKEYR('C:\Program Files\IBM\WebSphere MQ\Qmgrs\QM1\ssl\Mykey')
```

The key database file has the fully qualified file name: `C:\Program Files\IBM\WebSphere MQ\Qmgrs\QM1\ssl\Mykey.kdb`

Attention: Ensure that you do not include the `.kdb` extension in the file name on the SSLKEYR keyword, as the queue manager appends this extension automatically.

You can also alter your queue manager's attributes using the IBM MQ Explorer or PCF commands.

When you change the location of a queue manager's key database file, certificates are not transferred from the old location. If the key database file you are now accessing is a new key database file, you must populate it with the CA and personal certificates you need, as described in "Importing a personal certificate into a key repository on UNIX, Linux or Windows systems" on page 692.

Locating the key repository for an IBM MQ MQI client on UNIX, Linux and Windows systems.

The location of the key repository is given by the MQSSLKEYR variable, or specified in the MQCONNX call.

Examine the MQSSLKEYR environment variable to find the location of the key database file for your IBM MQ MQI client. For example:

```
echo $MQSSLKEYR
```

Also check your application, because the key database file name can also be set in an MQCONNX call, as described in"Specifying the key repository location for an IBM MQ MQI client on UNIX, Linux or Windows systems" on page 680. The value set in an MQCONNX call overrides the value of MQSSLKEYR.

Specifying the key repository location for an IBM MQ MQI client on UNIX, Linux or Windows systems

There is no default key repository for an IBM MQ MQI client. You can specify its location in either of two ways. Ensure that the key database file can be accessed only by intended users or administrators to prevent unauthorized copying to other systems.

You can specify the location of the key database file for your IBM MQ MQI client in two ways:

- Setting the MQSSLKEYR environment variable. For example, on UNIX and Linux systems:

  ```
  export MQSSLKEYR=/var/mqm/ssl/key
  ```

  The key database file has the fully-qualified file name:

  ```
  /var/mqm/ssl/key.kdb
  ```

  On Windows:

  ```
  set MQSSLKEYR=C:\Program Files\IBM\WebSphere MQ\ssl\key
  ```

  The key database file has the fully-qualified file name:

  ```
  C:\Program Files\IBM\WebSphere MQ\ssl\key.kdb
  ```

  Note: The `.kdb` extension is a mandatory part of the file name, but is not included as part of the value of the environment variable.
- Providing the path and stem name of the key database file in the *KeyRepository* field of the MQSCO structure when an application makes an MQCONNX call. For more information about using the MQSCO structure in MQCONNX, see Overview for MQSCO.

When changes to certificates or the certificate store become effective on UNIX, Linux or Windows systems.

When you change the certificates in a certificate store, or the location of the certificate store, the changes take effect depending on the type of channel and how the channel is running.

Changes to the certificates in the key database file and to the key repository attribute become effective in the following situations:

- When a new outbound single channel process first runs an SSL channel.
- When a new inbound TCP/IP single channel process first receives a request to start an SSL channel.
- When the MQSC command REFRESH SECURITY TYPE(SSL) is issued to refresh the Websphere MQ SSL environment.
- For client application processes, when the last SSL connection in the process is closed. The next SSL connection will pick up the certificate changes.
- For channels that run as threads of a process pooling process (amqrmppa), when the process pooling process is started or restarted and first runs an SSL channel. If the process pooling process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).
- For channels that run as threads of the channel initiator, when the channel initiator is started or restarted and first runs an SSL channel. If the channel initiator process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).
- For channels that run as threads of a TCP/IP listener, when the listener is started or restarted and first receives a request to start an SSL channel. If the listener has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).

You can also refresh the IBM MQ SSL environment using the IBM MQ Explorer or PCF commands.

Creating a self-signed personal certificate on UNIX, Linux, and Windows systems

You can create a self-signed certificate by using the **strmqikm** (iKeyman) GUI, or from the command line using **runmqckm** (iKeycmd) or **runmqakm** (GSKCapiCmd).

Note: IBM MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

For more information about why you might want to use self-signed certificates, see Using self-signed certificates for mutual authentication of two queue managers.

Not all digital certificates can be used with all CipherSpecs. Ensure that you create a certificate that is compatible with the CipherSpecs you need to use. IBM MQ supports three different types of CipherSpec. For details, see "Interoperability of Elliptic Curve and RSA CipherSpecs" on page 420 in the "Digital certificates and CipherSpec compatibility in IBM MQ" on page 419 topic. To use the Type 1 CipherSpecs (those with names beginning `ECDHE_ECDSA_`) you must use the **runmqakm** command to create the certificate and you must specify an Elliptic Curve ECDSA signature algorithm parameter; for example, **-sig_alg** `EC_ecdsa_with_SHA384`.

Using iKeyman

iKeyman does not provide a FIPS-compliant option. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Use the following procedure to obtain a self-signed certificate for your queue manager or IBM MQ MQI client:

1. Start the iKeyman GUI by using the **strmqikm** command .
2. From the **Key Database File** menu, click **Open**. The Open window displays.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file in which you want to save the certificate, for example `key.kdb`.
6. Click **OK**. The Password Prompt window displays.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is displayed in the **File Name** field.
8. From the **Create** menu, click **New Self-Signed Certificate**. The Create New Self-Signed Certificate window is displayed.
9. In the **Key Label** field, enter the certificate label. The label is either the value of the **CERTLABL** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client logon user ID appended, all in lowercase. See Digital certificate labels for details.
10. Type or select a value for any field in the **Distinguished name**, or any of the **Subject alternative name** fields.
11. For the remaining fields, either accept the default values, or type or select new values. For more information about Distinguished Names, see "Distinguished Names" on page 384.
12. Click **OK**. The **Personal Certificates** list shows the label of the self-signed personal certificate you created.

Using the command line

Use the following commands to create a self-signed personal certificate by using **iKeycmd** or **runmqakm**:

- Using **iKeycmd** on UNIX, Linux and Windows systems:

```
runmqckm -cert -create -db filename -pw password -label label
        -dn distinguished_name -size key_size
 -x509version version -expire days
 -sig_alg algorithm
```

Instead of -dn *distinguished_name*, you can use -san_dsname *DNS_names*, -san_emailaddr *email_addresses*, or -san_ipaddr *IP_addresses*.

- Using **runmqakm**:

```
runmqakm -cert -create -db filename -pw password -label label
        -dn distinguished_name -size key_size
 -x509version version -expire days

        -fips -sig_alg algorithm
```

-db *filename*	The fully qualified file name of a CMS key database.
-pw *password*	The password for the CMS key database.
-label *label*	The key label attached to the certificate. The label is either the value of the **CERTLABL** attribute, if it is set, or the default ibmwebspheremq with the name of the queue manager or the IBM MQ MQI client logon user ID appended, all in lowercase. See "Digital certificate labels, understanding the requirements" on page 400 for details.
-dn *distinguished_name*	The X.500 distinguished name enclosed in double quotation marks. At least one attribute is required. You can supply multiple OU or DC attributes. **Note:** The **iKeycmd** and **runmqakm** tools refer to the postal code attribute as POSTALCODE, not PC. Always specify POSTALCODE in the **-dn** parameter when you use these certificate management commands to request certificates with a postal code.
-size *key_size*	The key size. For **iKeycmd**, the value can be 512 or 1024. For runmqakm, the value can be 512, 1024, 2048 or 4096.
-x509version *version*	The version of X.509 certificate to create. The value can be 1, 2, or 3. The default is 3.
-expire *days*	The expiration time in days of the certificate. The default is 365 days for a certificate.
-fips	Specifies that the command is run in FIPS mode. Only the FIPS ICC component is used and this component must be successfully initialized in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.
-sig_alg	For runmqakm, the hashing algorithm used during the creation of a self-signed certificate. This hashing algorithm is used to create the signature associated with the newly created self-signed certificate. The value can be md5, MD5_WITH_RSA, MD5WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, sha1, SHA1WithDSA, SHA1WithECDSA, SHA1WithRSA, sha224, SHA224_WITH_RSA, SHA224WithDSA, SHA224WithECDSA, SHA224WithRSA, sha256, SHA256_WITH_RSA, SHA256WithDSA, SHA256WithECDSA, SHA256WithRSA, SHA2WithRSA, sha384, SHA384_WITH_RSA, SHA384WithECDSA, SHA384WithRSA, sha512, SHA512_WITH_RSA, SHA512WithECDSA, SHA512WithRSA, SHAWithDSA, SHAWithRSA, EC_ecdsa_with_SHA1, EC_ecdsa_with_SHA224, EC_ecdsa_with_SHA256, EC_ecdsa_with_SHA384, or EC_ecdsa_with_SHA512. The default value is SHA1WithRSA.
-sig_alg	For iKeycmd, the asymmetric signature algorithm used for the creation of the entry's key pair. The value can be MD2_WITH_RSA, MD2WithRSA, MD5_WITH_RSA, MD5WithRSA, SHA1WithDSA, SHA1WithRSA, SHA256_WITH_RSA, SHA256WithRSA, SHA2WithRSA, SHA384_WITH_RSA, SHA384WithRSA, SHA512_WITH_RSA, SHA512WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, SHAWithDSA, or SHAWithRSA. The default value is SHA1WithRSA.
-san_dnsname *DNS_names*	A comma- or space-delimited list of DNS names for the entry being created.
-san_emailaddr *email_addresses*	A comma- or space-delimited list of email addresses for the entry being created.
-san_ipaddr *IP_addresses*	A comma- or space-delimited list of IP addresses for the entry being created.

Requesting a personal certificate on UNIX, Linux, and Windows systems

distributed

You can request a personal certificate by using the **strmqikm** (iKeyman) GUI, or from the command line using the **runmqckm** (iKeycmd) or **runmqakm** (GSKCapiCmd) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

About this task

You can request a personal certificate using the iKeyman GUI, or from the command line, subject to the following considerations:

- IBM MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.
- The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.
- Not all digital certificates can be used with all CipherSpecs. Ensure that you request a certificate that is compatible with the CipherSpecs you need to use. IBM MQ supports three different types of CipherSpec. For details, see “Interoperability of Elliptic Curve and RSA CipherSpecs” on page 420 in the “Digital certificates and CipherSpec compatibility in IBM MQ” on page 419 topic.
- To use the Type 1 CipherSpecs (with names beginning `ECDHE_ECDSA_`) you must use the **runmqakm** command to request the certificate and you must specify an Elliptic Curve ECDSA signature algorithm parameter; for example, **-sig_alg** `EC_ecdsa_with_SHA384`.
- Only the runmqakm command provides a FIPS-compliant option.
- If you are using cryptographic hardware, see “Requesting a personal certificate for your PKCS #11 hardware” on page 700.

Using the iKeyman user interface:
About this task

iKeyman does not provide a FIPS-compliant option. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Procedure

Complete the following steps to apply for a personal certificate, by using the iKeyman user interface:

1. Start the iKeyman user interface by using the **strmqikm** command.
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file from which you want to generate the request; for example, `key.kdb`.
6. Click **Open**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is shown in the **File Name** field.
8. From the **Create** menu, click **New Certificate Request**. The Create New Key and Certificate Request window opens.
9. In the **Key Label** field, enter the certificate label. The label is either the value of the **CERTLABL** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client logon user ID appended, all in lowercase. See Digital certificate labels for details.

10. Type or select a value for any field in the **Distinguished name** field, or any of the **Subject alternative name** fields. For the remaining fields, either accept the default values, or type or select new values. For more information about Distinguished Names, see "Distinguished Names" on page 384.
11. In the **Enter the name of a file in which to store the certificate request** field, either accept the default `certreq.arm`, or type a new value with a full path.
12. Click **OK**. A confirmation window is displayed.
13. Click **OK**. The **Personal Certificate Requests** list shows the label of the new personal certificate request you created. The certificate request is stored in the file you chose in step 11.
14. Request the new personal certificate either by sending the file to a certificate authority (CA), or by copying the file into the request form on the website for the CA.

Using the command line:
Procedure

Request a personal certificate by using either the **runmqckm** (iKeycmd) or **runmqakm** (GSKCapiCmd) command.

- Using **runmqckm**:

  ```
  runmqckm -certreq -create -db filename -pw password -label label
          -dn distinguished_name -size key_size
   -file filename -sig_alg algorithm
  ```

  Instead of `-dn` *distinguished_name*, you can use `-san_dsname` *DNS_names*, `-san_emailaddr` *email_addresses*, or `-san_ipaddr` *IP_addresses*.
- Using **runmqakm**:

  ```
  runmqakm -certreq -create -db filename -pw password -label label
          -dn distinguished_name -size key_size
   -file filename -fips
          -sig_alg algorithm
  ```

where:

-db *filename*
Specifies the fully qualified file name of a CMS key database.

-pw *password*
Specifies the password for the CMS key database.

-label *label*
Specifies the key label attached to the certificate. The label is either the value of the **CERTLABL** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or the IBM MQ MQI client logon user ID appended, all in lowercase. See "Digital certificate labels, understanding the requirements" on page 400 for details.

-dn *distinguished_name*
Specifies the X.500 distinguished name enclosed in double quotation marks. At least one attribute is required. You can supply multiple OU and DC attributes.

Note: The **runmqckm** and **runmqakm** tools refer to the postal code attribute as `POSTALCODE`, not `PC`. Always specify `POSTALCODE` in the **-dn** parameter when you use these certificate management commands to request certificates with a postal code.

-size *key_size*
Specifies the key size. If you are using **runmqckm**, the value can be `512` or `1024`. If you are using **runmqakm**, the value can be `512`, `1024`, or `2048`.

-file *filename*
Specifies the file name for the certificate request.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-sig_alg
For **runmqckm**, specifies the asymmetric signature algorithm used for the creation of the entry's key pair. The value can be MD2_WITH_RSA, MD2WithRSA, MD5_WITH_RSA, MD5WithRSA, SHA1WithDSA, SHA1WithRSA, SHA256_WITH_RSA, SHA256WithRSA, SHA2WithRSA, SHA384_WITH_RSA, SHA384WithRSA, SHA512_WITH_RSA, SHA512WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, SHAWithDSA, or SHAWithRSA. The default value is SHA1WithRSA

-sig_alg
For **runmqakm**, specifies the hashing algorithm used during the creation of a certificate request. This hashing algorithm is used to create the signature associated with the newly created certificate request. The value can be md5, MD5_WITH_RSA, MD5WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, sha1, SHA1WithDSA, SHA1WithECDSA, SHA1WithRSA, sha224, SHA224_WITH_RSA, SHA224WithDSA, SHA224WithECDSA, SHA224WithRSA, sha256, SHA256_WITH_RSA, SHA256WithDSA, SHA256WithECDSA, SHA256WithRSA, SHA2WithRSA, sha384, SHA384_WITH_RSA, SHA384WithECDSA, SHA384WithRSA, sha512, SHA512_WITH_RSA, SHA512WithECDSA, SHA512WithRSA, SHAWithDSA, SHAWithRSA, EC_ecdsa_with_SHA1, EC_ecdsa_with_SHA224, EC_ecdsa_with_SHA256, EC_ecdsa_with_SHA384, or EC_ecdsa_with_SHA512. The default value is SHA1WithRSA.

-san_dnsname ***DNS_names***
Specifies a comma-delimited or space-delimited list of DNS names for the entry being created.

-san_emailaddr ***email_addresses***
Specifies a comma-delimited or space-delimited list of email addresses for the entry being created.

-san_ipaddr ***IP_addresses***
Specifies a comma-delimited or space-delimited list of IP addresses for the entry being created.

What to do next

Submit a certificate request to a CA. See "Receiving personal certificates into a key repository on UNIX, Linux and Windows systems" on page 687 for further information.

Renewing an existing personal certificate on UNIX, Linux, and Windows systems

You can renew a personal certificate by using the **strmqikm** (iKeyman) GUI, or from the command line using the **runmqckm** (iKeycmd) or **runmqakm** (GSKCapiCmd) commands.

About this task

If you have a requirement to use larger key sizes for your personal certificates, you cannot renew an existing certificate. You must replace your existing key by following the steps described in "Requesting a personal certificate on UNIX, Linux, and Windows systems" on page 683 to create a new certificate request that uses the key sizes you require.

A personal certificate has an expiry date, after which the certificate can no longer be used. This task explains how to renew an existing personal certificate before it expires.

Using the iKeyman user interface:
About this task

iKeyman does not provide a FIPS-compliant option. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Procedure

Complete the following steps to apply for a personal certificate, by using the iKeyman user interface:

1. Start the iKeyman user interface by using the **strmqikm** command on UNIX, Linux, and Windows systems.
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file from which you want to generate the request; for example, `key.kdb`.
6. Click **Open**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is shown in the **File Name** field.
8. Select **Personal Certificates** from the drop down selection menu, and select the certificate from the list that you want to renew.
9. Click the **Recreate Request...** button. A window opens for you to enter the file name and file location information.
10. In the **file name** field, either accept the default `certreq.arm`, or type a new value, including the full file path.
11. Click **OK**. The certificate request is stored in the file you selected in step 9.
12. Request the new personal certificate either by sending the file to a certificate authority (CA), or by copying the file into the request form on the website for the CA.

Using the command line:
Procedure

Use the following commands to request a personal certificate by using either the **iKeycmd** or **runmqakm** command:

- Using **iKeycmd** on UNIX, Linux, and Windows systems:
```
runmqckm -certreq -recreate -db filename -pw password -label label
 -target filename
```
- Using runmqakm:
```
runmqakm -certreq -recreate -db filename -pw password -label label
-target filename
```

where:

-db *filename*
Specifies the fully qualified file name of a CMS key database.

-pw *password*
Specifies the password for the CMS key database.

-target *filename*
Specifies the file name for the certificate request.

What to do next

Once you have received the signed personal certificate from the certificate authority, you can add it to your key database using the steps described in "Receiving personal certificates into a key repository on UNIX, Linux and Windows systems."

Receiving personal certificates into a key repository on UNIX, Linux and Windows systems

Use this procedure to receive a personal certificate into the key database file. The key repository must be the same repository where you created the certificate request.

After the CA sends you a new personal certificate, you add it to the key database file from which you generated the new certificate request . If the CA sends the certificate as part of an email message, copy the certificate into a separate file.

Using iKeyman

If you need to manage SSL certificates in a way that is FIPS compliant, use the runmqakm command. iKeyman does not provide a FIPS-compliant option.

Ensure that the certificate file to be imported has write permission for the current user, and then use the following procedure for either a queue manager or an IBM MQ MQI client to receive a personal certificate into the key database file:

1. Start the iKeyman GUI using the **`strmqikm`** command (on Windows UNIX and Linux).
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file to which you want to add the certificate, for example `key.kdb`.
6. Click **Open**, and then click **OK**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is displayed in the **File Name** field. Select the **Personal Certificates** view.
8. Click **Receive**. The Receive Certificate from a File window opens.
9. Type the certificate file name and location for the new personal certificate, or click **Browse** to select the name and location.
10. Click **OK**. If you already have a personal certificate in your key database, a window opens, asking if you want to set the key you are adding as the default key in the database.
11. Click **Yes** or **No**. The Enter a Label window opens.
12. Click **OK**. The **Personal Certificates** field shows the label of the new personal certificate you added.

Using the command line

Use the following commands to add a personal certificate to a key database file using iKeycmd, or runmqakm:

- On UNIX, Linux and Windows, issue the following command:

```
runmqckm -cert -receive -file filename -db filename -pw password
-format ascii
```

where:

`-file` *filename*	is the fully qualified file name of the file containing the personal certificate.
`-db` *filename*	is the fully qualified file name of a CMS key database.
`-pw` *password*	is the password for the CMS key database.
`-format` *ascii*	is the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for Binary DER data. The default is `ascii`.
`-fips`	specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

If you are using cryptographic hardware, refer to "Importing a personal certificate to your PKCS #11 hardware" on page 702.

Extracting a CA certificate from a key repository

Follow this procedure to extract a CA certificate.

Using iKeyman

If you need to manage SSL certificates in a way that is FIPS compliant, use the runmqakm command. iKeyman does not provide a FIPS-compliant option.

Perform the following steps on the machine from which you want to extract the CA certificate:

1. Start the iKeyman GUI using the **strmqikm** command..
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file from which you want to extract, for example `key.kdb`.
6. Click **Open**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is displayed in the **File Name** field.
8. In the **Key database content** field, select **Signer Certificates** and select the certificate you want to extract.
9. Click **Extract**. The Extract a Certificate to a File window opens.
10. Select the **Data type** of the certificate, for example **Base64-encoded ASCII data** for a file with the `.arm` extension.
11. Type the certificate file name and location where you want to store the certificate, or click **Browse** to select the name and location.
12. Click **OK**. The certificate is written to the file you specified.

Using the command line

Use the following commands to extract a CA certificate using iKeycmd :

- On UNIX, Linux and Windows:

```
runmqckm -cert -extract -db filename -pw password -label label -target filename
        -format ascii
```

where:

`-db` *filename*	is the fully qualified path name of a CMS key database.
`-pw` *password*	is the password for the CMS key database.
`-label` *label*	is the label attached to the certificate.
`-target` *filename*	is the name of the destination file.
`-format` *ascii*	is the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for Binary DER data. The default is `ascii`.
`-fips`	specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

Extracting the public part of a self-signed certificate from a key repository on UNIX, Linux and Windows systems

Follow this procedure to extract the public part of a self-signed certificate.

Using iKeyman

If you need to manage SSL certificates in a way that is FIPS compliant, use the runmqakm command. iKeyman does not provide a FIPS-compliant option.

Perform the following steps on the machine from which you want to extract the public part of a self-signed certificate:

1. Start the iKeyman GUI using the **strmqikm** command (on UNIX, Linux and Windows).
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file from which you want to extract the certificate, for example `key.kdb`.
6. Click **OK**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is displayed in the **File Name** field.
8. In the **Key database content** field, select **Personal Certificates** and select the certificate.
9. Click **Extract certificate**. The Extract a Certificate to a File window opens.
10. Select the **Data type** of the certificate, for example **Base64-encoded ASCII data** for a file with the `.arm` extension.
11. Type the certificate file name and location where you want to store the certificate, or click **Browse** to select the name and location.
12. Click **OK**. The certificate is written to the file you specified. Note that when you extract (rather than export) a certificate, only the public part of the certificate is included, so a password is not required.

Using the command line

Use the following commands to extract the public part of a self-signed certificate using iKeycmd or runmqakm:

- On UNIX, Linux and Windows:
  ```
  runmqckm -cert -extract -db filename -pw password -label label -target filename
          -format ascii
  ```
- Using runmqakm:
  ```
  runmqakm -cert -extract -db filename -pw password -label label
             -target filename -format ascii -fips
  ```

where:

`-db` *filename*	is the fully qualified path name of a CMS key database.
`-pw` *password*	is the password for the CMS key database.
`-label` *label*	is the label attached to the certificate.
`-target` *filename*	is the name of the destination file.
`-format` *ascii*	is the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for Binary DER data. The default is `ascii`.
`-fips`	specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

Adding a CA certificate (or the public part of a self-signed certificate) into a key repository, on UNIX, Linux or Windows systems

Follow this procedure to add a CA certificate or the public part of a self-signed certificate to the key repository.

If the certificate that you want to add is in a certificate chain, you must also add all the certificates that are above it in the chain. You must add the certificates in strictly descending order starting from the root, followed by the CA certificate immediately below it in the chain, and so on.

Where the following instructions refer to a CA certificate, they also apply to the public part of a self-signed certificate.

Note: You must ensure that the certificate is in ASCII (UTF-8) or binary (DER) encoding, because IBM Global Secure Toolkit (GSKit) does not support certificates with other types of encoding.

Using iKeyman

If you need to manage SSL certificates in a way that is FIPS compliant, use the runmqakm command. iKeyman does not provide a FIPS-compliant option.

Perform the following steps on the machine on which you want to add the CA certificate:

1. Start the iKeyman GUI using the **strmqikm** command (on UNIX, Linux and Windows systems).
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file to which you want to add the certificate, for example `key.kdb`.
6. Click **OK**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file displays in the **File Name** field.
8. In the **Key database content** field, select **Signer Certificates**.
9. Click **Add**. The Add CA's Certificate from a File window opens.
10. Type the certificate file name and location where the certificate is stored, or click **Browse** to select the name and location.
11. Click **OK**. The Enter a Label window opens.
12. In the Enter a Label window, type the name of the certificate.
13. Click **OK**. The certificate is added to the key database.

Using the command line

Use the following commands to add a CA certificate using iKeycmd or runmqakm:

- On UNIX, Linux and Windows, issue the following command:

```
runmqckm -cert -add -db filename -pw password -label label -file filename
        -format ascii
```

where:

-db *filename*	is the fully qualified path name of the CMS key database.
-pw *password*	is the password for the CMS key database.
-label *label*	is the label attached to the certificate.
-file *filename*	is the name of the file containing the certificate.
-format *ascii*	is the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for Binary DER data. The default is `ascii`.

Exporting a personal certificate from a key repository

Follow this procedure to exporting a personal certificate.

Using iKeyman

If you need to manage SSL certificates in a way that is FIPS compliant, use the runmqakm command. iKeyman does not provide a FIPS-compliant option.

Perform the following steps on the machine from which you want to export the personal certificate:

1. Start the iKeyman GUI using the **`strmqikm`** command (on Windows UNIX and Linux).
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file from which you want to export the certificate, for example `key.kdb`.
6. Click **Open**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is displayed in the **File Name** field.
8. In the **Key database content** field, select **Personal Certificates** and select the certificate you want to export.
9. Click **Export/Import**. The Export/Import key window opens.
10. Select **Export Key**.
11. Select the **Key file type** of the certificate you want to export, for example **PKCS12**.
12. Type the file name and location to which you want to export the certificate, or click **Browse** to select the name and location.
13. Click **OK**. The Password Prompt window opens. Note that when you export (rather than extract) a certificate, both the public and private parts of the certificate are included. This is why the exported file is protected by a password. When you extract a certificate, only the public part of the certificate is included, so a password is not required.
14. Type a password in the **Password** field, and type it again in the **Confirm Password** field.
15. Click **OK**. The certificate is exported to the file you specified.

Using the command line

Use the following commands to export a personal certificate using iKeycmd:

- On UNIX, Linux and Windows:

```
runmqckm -cert -export -db filename -pw password -label label -type cms
        -target filename -target_pw password -target_type pkcs12
```

where:

`-db` *filename*	is the fully qualified path name of the CMS key database.
`-fips`	specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.
`-pw` *password*	is the password for the CMS key database.
`-label` *label*	is the label attached to the certificate.
`-type` *cms*	is the type of the database.
`-target` *filename*	is the fully qualified path name of the destination file.
`-target_pw` *password*	is the password for encrypting the certificate.
`-target_type` *pkcs12*	is the type of the certificate.

Importing a personal certificate into a key repository on UNIX, Linux or Windows systems

Follow this procedure to import a personal certificate

Before importing a personal certificate in PKCS #12 format into the key database file, you must first add the full valid chain of issuing CA certificates to the key database file (see "Adding a CA certificate (or the public part of a self-signed certificate) into a key repository, on UNIX, Linux or Windows systems" on page 690).

PKCS #12 files should be considered temporary and deleted after use.

Using iKeyman

If you need to manage SSL certificates in a way that is FIPS-compliant, use the runmqakm command. iKeyman does not provide a FIPS-compliant option.

Perform the following steps on the machine to which you want to import the personal certificate:

1. Start the iKeyman GUI using the **strmqikm** command .
2. From the **Key Database File** menu, click **Open**. The Open window displays.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file to which you want to add the certificate, for example `key.kdb`.
6. Click **Open**. The Password Prompt window displays.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file displays in the **File Name** field.
8. In the **Key database content** field, select **Personal Certificates**.
9. If there are certificates in the Personal Certificates view, follow these steps:
   a. Click **Export/Import**. The Export/Import key window is displayed.
   b. Select **Import Key**.
10. If there are no certificates in the Personal Certificates view, click **Import**.
11. Select the **Key file type** of the certificate you want to import, for example PKCS12.
12. Type the certificate file name and location where the certificate is stored, or click **Browse** to select the name and location.
13. Click **OK**. The Password Prompt window displays.
14. In the **Password** field, type the password used when the certificate was exported.
15. Click **OK**. The Change Labels window is displayed. You can change the labels of certificates being imported if, for example, a certificate with the same label already exists in the target key database. Changing certificate labels has no effect on certificate chain validation. To associate the certificate with a particular queue manager or IBM MQ MQI client, IBM MQ uses either the value of the

CERTLABL attribute, if it is set, or the default ibmwebspheremq with the name of the queue manager or IBM MQ MQI client user logon ID appended, all in lowercase. See Digital certificate labels for details.

16. To change a label, select the required label from the **Select a label to change** list. The label is copied into the **Enter a new label** entry field. Replace the label text with that of the new label and click **Apply**.
17. The text in the **Enter a new label** entry field is copied back into the **Select a label to change** field, replacing the originally selected label and so relabelling the corresponding certificate.
18. When you have changed all the labels that needed to be changed, click **OK**. The Change Labels window closes, and the original IBM Key Management window reappears with the **Personal Certificates** and **Signer Certificates** fields updated with the correctly labeled certificates.
19. The certificate is imported to the target key database.

Using the command line

To import a personal certificate using iKeycmd, use the following commands:

- On UNIX, Linux and Windows:

```
runmqckm -cert -import -file filename -pw password -type pkcs12 -target filename
-target_pw password -target_type cms -label label
```

where:

-file *filename*	is the fully qualified file name of the file containing the PKCS #12 certificate.
-pw *password*	is the password for the PKCS #12 certificate.
-type *pkcs12*	is the type of the file.
-target *filename*	is the name of the destination CMS key database.
-target_pw *password*	is the password for the CMS key database.
-target_type *cms*	is the type of the database specified by -target
-label *label*	is the label of the certificate to import from the source key database.
-new_label *label*	is the label that the certificate will be assigned in the target database. If you omit -new_label option, the default is to use the same as the -label option.
-fips	specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

iKeycmd does not provide a command to change certificate labels directly. Use the following steps to change a certificate label:

1. Export the certificate to a PKCS #12 file using the **-cert -export** command. Specify the existing certificate label for the -label option.
2. Remove the existing copy of the certificate from the original key database using the **-cert -delete** command.
3. Import the certificate from the PKCS #12 file using the **-cert -import** command. Specify the old label for the -label option and the required new label for the -new_label option. The certificate will be imported back into the key database with the required label.

Importing a personal certificate from a Microsoft.pfx file

Follow this procedure to import from a Microsoft.pfx file.

A .pfx file can contain two certificates relating to the same key. One is a personal or site certificate (containing both a public and private key). The other is a CA (signer) certificate (containing only a public key). These certificates cannot coexist in the same CMS key database file, so only one of them can be imported. Also, the "friendly name" or label is attached to only the signer certificate.

The personal certificate is identified by a system generated Unique User Identifier (UUID). This section shows the import of a personal certificate from a pfx file while labeling it with the friendly name previously assigned to the CA (signer) certificate. The issuing CA (signer) certificates should already be added to the target key database. Note that PKCS#12 files should be considered temporary and deleted after use.

Follow these steps to import a personal certificate from a source pfx key database:

1. Start the iKeyman GUI using the `strmqikm` command (on Linux, UNIX or Windows). The IBM Key Management window is displayed.
2. From the **Key Database File** menu, click **Open**. The Open window is displayed.
3. Select a key database type of **PKCS12**.
4. **You are recommended to take a backup of the pfx database before performing this step.** Select the pfx key database that you want to import. Click **Open**. The Password Prompt window is displayed.
5. Enter the key database password and click **OK**. The IBM Key Management window is displayed. The title bar shows the name of the selected pfx key database file, indicating that the file is open and ready.
6. Select **Signer Certificates** from the list. The "friendly name" of the required certificate is displayed as a label in the Signer Certificates panel.
7. Select the label entry and click **Delete** to remove the signer certificate. The Confirm window is displayed.
8. Click **Yes**. The selected label is no longer displayed in the Signer Certificates panel.
9. Repeat steps 6, 7, and 8 for all the signer certificates.
10. From the **Key Database File** menu, click **Open**. The Open window is displayed.
11. Select the target key CMS database which the pfx file is being imported into. Click **Open**. The Password Prompt window is displayed.
12. Enter the key database password and click **OK**. The IBM Key Management window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
13. Select **Personal Certificates** from the list.
14. If there are certificates in the Personal Certificates view, follow these steps:
    a. Click **Export/Import key**. The Export/Import key window is displayed.
    b. Select **Import** from Choose Action Type.
15. If there are no certificates in the Personal Certificates view, click **Import**.
16. Select the PKCS12 file.
17. Enter the name of the pfx file as used in Step 4. Click **OK**. The Password Prompt window is displayed.
18. Specify the same password that you specified when you deleted the signer certificate. Click **OK**.
19. The Change Labels window is displayed (as there should be only a single certificate available for import). The label of the certificate should be a UUID which has a format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.
20. To change the label select the UUID from the **Select a label to change:** panel. The label will be replicated into the **Enter a new label:** field. Replace the label text with that of the friendly name that

was deleted in Step 7 and click **Apply**. The friendly name must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client user logon ID appended, all in lowercase. See Digital certificate labels for details.

21. Click **OK**. The Change Labels window is now removed and the original IBM Key Management window reappears with the Personal Certificates and Signer Certificates panels updated with the correctly labeled personal certificate.
22. The pfx personal certificate is now imported to the (target) database.

It is not possible to change a certificate label using iKeycmd or runmqakm.

Using the command line

To import a personal certificate using iKeycmd, use the following commands:

- On UNIX, Linux and Windows:

```
runmqckm -cert -import -file filename -pw password -type pkcs12 -target filename
-target_pw password -target_type cms -label label -pfx
```

To import a personal certificate using **runmqakm**, use the following command:

```
runmqakm -cert -import -file filename -pw password -type pkcs12 -target filename
-target_pw password -target_type cms -label label -fips -pfx
```

where:

`-file` *filename*	is the fully qualified file name of the file containing the PKCS #12 certificate.
`-pw` *password*	is the password for the PKCS #12 certificate.
`-type` *pkcs12*	is the type of the file.
`-target` *filename*	is the name of the destination CMS key database.
`-target_pw` *password*	is the password for the CMS key database.
`-target_type` *cms*	is the type of the database specified by `-target`
`-label` *label*	is the label of the certificate to import from the source key database.
`-new_label` *label*	is the label that the certificate will be assigned in the target database. If you omit `-new_label` option, the default is to use the same as the `-label` option.
`-fips`	specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.
`-pfx`	indicates PFX file format.

iKeycmd does not provide a command to change certificate labels directly. Use the following steps to change a certificate label:

1. Export the certificate to a PKCS #12 file using the **-cert -export** command. Specify the existing certificate label for the `-label` option.
2. Remove the existing copy of the certificate from the original key database using the **-cert -delete** command.
3. Import the certificate from the PKCS #12 file using the **-cert -import** command. Specify the old label for the `-label` option and the required new label for the `-new_label` option. The certificate will be imported back into the key database with the required label.

Importing a personal certificate from a PKCS #7 file

The iKeyman and iKeycmd tools do not support PKCS #7 (`.p7b`) files. Use the runmqckm tool to import certificates from a PKCS #7 file.

Use the following command to add a CA certificate from a PKCS #7 file:

```
runmqckm -cert -add -db filename -pw password -type cms -file filename
-label label
```

`-db` *filename*	is the fully qualified file name of the CMS key database.
`-pw` *password*	is the password for the key database.
`-type` *cms*	is the type of the key database.
`-file` *filename*	is the name of the PKCS #7 file.
`-label` *label*	is the label that the certificate is assigned in the target database. The first certificate takes the label given. All other certificates, if present, are labeled with their subject name.

Use the following command to import a personal certificate from a PKCS #7 file:

```
runmqckm -cert -import -db filename -pw password -type pkcs7 -target filename
-target_pw password -target_type cms -label label -new_label label
```

`-db` *filename*	is the fully qualified file name of the file containing the PKCS #7 certificate.
`-pw` *password*	is the password for the PKCS #7 certificate.
`-type` *pkcs7*	is the type of the file.
`-target` *filename*	is the name of the destination key database.
`-target_pw` *password*	is the password for the destination key database.
`-target_type` *cms*	is the type of the database specified by `-target`
`-label` *label*	is the label of the certificate that is to be imported.
`-new_label` *label*	is the label that the certificate will be assigned in the target database. If you omit the `-new_label` option, the default is to use the same as the `-label` option.

Deleting a certificate from a key repository on UNIX, Linux or Windows systems

Use this procedure to remove personal or CA certificates.

Using iKeyman

If you need to manage SSL certificates in a way that is FIPS compliant, use the runmqakm command. iKeyman does not provide a FIPS-compliant option.

1. Start the iKeyman GUI using the **`strmqikm`** command (on UNIX, Linux and Windows systems).
2. From the **Key Database File** menu, click **Open**. The Open window opens.
3. Click **Key database type** and select **CMS** (Certificate Management System).
4. Click **Browse** to navigate to the directory that contains the key database files.
5. Select the key database file from which you want to delete the certificate, for example `key.kdb`.
6. Click **Open**. The Password Prompt window opens.
7. Type the password you set when you created the key database and click **OK**. The name of your key database file is displayed in the **File Name** field.
8. From the drop down list, select **Personal Certificates** or **Signer Certificates**
9. Select the certificate you want to delete.
10. If you do not already have a copy of the certificate and you want to save it, click **Export/Import** and export it (see "Exporting a personal certificate from a key repository" on page 691).
11. With the certificate selected, click **Delete**. The Confirm window opens.

12. Click **Yes**. The **Personal Certificates** field no longer shows the label of the certificate you deleted.

Using the command line

Use the following commands to delete a certificate using iKeycmd or runmqakm:

- On UNIX, Linux and Windows:

```
runmqckm -cert -delete -db filename -pw password -label label
```

where:

`-db` *filename*	is the fully qualified file name of a CMS key database.
`-pw` *password*	is the password for the CMS key database.
`-label` *label*	is the label attached to the personal certificate.
`-fips`	specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

Generating strong passwords for key repository protection

You can generate strong passwords for key repository protection using the **runmqakm** (GSKCapiCmd) command.

You can use the **runmqakm** command with the following parameters to generate a strong password:

```
runmqakm -random -create -length 14 -strong -fips
```

When using the generated password on the **-pw** parameter of subsequent certificate administration commands, always place double quotation marks around the password. On UNIX and Linux systems, you must also use a backslash character to escape the following characters if they appear in the password string:

```
!  \  "  '
```

When entering the password in response to a prompt from **runmqckm**, **runmqakm** or the iKeyman GUI then it is not necessary to quote or escape the password. It is not necessary because the operating system shell does not affect data entry in these cases.

Configuring for cryptographic hardware on UNIX, Linux or Windows systems

You can configure cryptographic hardware for a queue manager or client in a number of ways.

You can configure cryptographic hardware for a queue manager on UNIX, Linux or Windows systems using either of the following methods:

- Use the ALTER QMGR MQSC command with the SSLCRYP parameter, as described in ALTER QMGR.
- Use IBM MQ Explorer to configure the cryptographic hardware on your UNIX, Linux or Windows system. For more information, refer to the online help.

You can configure cryptographic hardware for an IBM MQ client on UNIX, Linux or Windows systems using either of the following methods:

- Set the MQSSLCRYP environment variable. The permitted values for MQSSLCRYP are the same as for the SSLCRYP parameter, as described in ALTER QMGR. If you use the GSK_PCS11 version of the SSLCRYP parameter, the PKCS #11 token label must be specified entirely in lower-case.
- Set the **CryptoHardware** field of the SSL configuration options structure, MQSCO, on an MQCONNX call. For more information, see Overview for MQSCO.

If you have configured cryptographic hardware which uses the PKCS #11 interface using any of these methods, you must store the personal certificate for use on your channels in the key database file for the cryptographic token you have configured. This is described in "Managing certificates on PKCS #11 hardware."

Managing certificates on PKCS #11 hardware:

You can manage digital certificates on cryptographic hardware that supports the PKCS #11 interface.

About this task

You must create a key database to prepare the IBM MQ environment, even if you do not intend to store certificate authority (CA) certificates in it, but will store all your certificates on your cryptographic hardware. A key database is necessary for the queue manager to reference in its SSLKEYR field, or for the client application to reference in the MQSSLKEYR environment variable. This key database is also required if you are creating a certificate request.

Procedure

- To create a key database by using the iKeyman user interface, complete the following steps:
  1. On UNIX and Linux systems, log in as the root user. On Windows systems, log in as Administrator or as a member of the MQM group.
  2. Start the iKeyman user interface by running the **strmqikm** command.
  3. Click **Key Database File** > **Open**.
  4. Click **Key database type** and select **PKCS11Direct**.
  5. In the **File Name** field, type the name of the module for managing your cryptographic hardware; for example, `PKCS11_API.so`.

     If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.
  6. In the **Location** field, enter the path:
     - On UNIX and Linux systems, this might be `/usr/lib/pksc11`, for example.
     - On Windows systems, you can type the library name; for example, `cryptoki`.

     Click **OK**. The Open Cryptographic Token window opens.
  7. In the **Cryptographic Token Password** field, type the password that you set when you configured the cryptographic hardware.
  8. If your cryptographic hardware has the capacity to hold the signer certificates required to receive or import a personal certificate, clear both secondary key database check boxes and continue from step 12 on page 699. If you require a secondary CMS key database to hold the signer certificates, select either **Open existing secondary key database file** or **Create new secondary key database file**.
  9. In the **File Name** field, type a file name. This field already contains the text `key.kdb`. If your stem name is `key`, leave this field unchanged. If you specified a different stem name, replace `key` with your stem name. You must not change the `.kdb` suffix.
  10. In the **Location** field, type the path, for example:
      - For a queue manager: `/var/mqm/qmgrs/QM1/ssl`
      - For an IBM MQ MQI client: `/var/mqm/ssl`

      Click **OK**. The Password Prompt window opens.
  11. Enter a password.
      - If you selected **Open existing secondary key database file** in step 8, type a password in the **Password** field.
      - If you selected **Create new secondary key database file** in step 8:

   a. Type a password in the **Password** field, and type it again in the **Confirm Password** field.
   b. Select **Stash the password to a file**. Note that if you do not stash the password, attempts to start SSL channels fail because they cannot obtain the password required to access the key database file.
   c. Click **OK**. A window opens, confirming that the password is in file `key.sth` (unless you specified a different stem name).
12. Click **OK**. The Key database content frame displays.

- To create a key database by using the command line, use either of the following commands:
  - On UNIX, Linux, and Windows systems:

    ```
    runmqckm -keydb -create -db filename -pw password -type cms -stash
    ```
  - Using runmqakm:

    ```
    runmqakm -keydb -create -db filename -pw password -type cms
    -stash -fips -strong
    ```

  where:

  -db *filename*
  Specifies the fully qualified file name of a CMS key database, and must have a file extension of `.kdb`.

  -pw *password*
  Specifies the password for the CMS key database.

  -type *cms*
  Specifies the type of database. (For IBM MQ, it must be `cms`.)

  -stash
  Saves the key database password to a file.

  -fips
  Specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

  -strong
  Checks that the password entered satisfies the minimum requirements for password strength. The minimum requirements for a password are as follows:
  - The password must be a minimum length of 14 characters.
  - The password must contain a minimum of one lowercase character, one uppercase character, and one digit or special character. Special characters include the asterisk (*), the dollar sign ($), the number sign (#), and the percent sign (%). A space is classified as a special character.
  - Each character can occur a maximum of three times in a password.
  - A maximum of two consecutive characters in the password can be identical.
  - All characters are in the standard ASCII printable character set, within the range 0x20 - 0x7E.

Requesting a personal certificate for your PKCS #11 hardware:

Use this procedure for either a queue manager or an IBM MQ MQI client to request a personal certificate for your cryptographic hardware.

Using the iKeyman user interface:
About this task

Note: IBM MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

Procedure

To request a personal certificate from the iKeyman user interface, complete the following steps:
1. Complete the steps to work with your cryptographic hardware. See "Managing certificates on PKCS #11 hardware" on page 698.
2. From the **Create** menu, click **New Certificate Request**. The Create New Key and Certificate Request window opens.
3. In the **Key Label** field, enter the certificate label. The label is either the value of the **`CERTLABL`** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client logon user ID appended, all in lowercase. See Digital certificate labels for details.
4. Enter values for **Common Name** and **Organization**, and select a **Country**. For the remaining optional fields, either accept the default values, or type or select new values. Note that you can supply only one name in the **Organizational Unit** field. For more information about these fields, see "Distinguished Names" on page 384.
5. In the **Enter the name of a file in which to store the certificate request** field, either accept the default `certreq.arm`, or type a new value with a full path.
6. Click **OK**. A confirmation window opens.
7. Click **OK**. The **Personal Certificate Requests** list shows the label of the new personal certificate request you created. The certificate request is stored in the file you chose in step 5.
8. Request the new personal certificate either by sending the file to a certificate authority (CA), or by copying the file into the request form on the website for the CA.

Using the command line:
Procedure

Request a personal certificate by using either the **`runmqckm`** (iKeycmd) or **`runmqakm`** (GSKCapiCmd) command.

- Using **`runmqckm`**:

  ```
  runmqckm -certreq -create -db filename -pw password -label label
          -dn distinguished_name -size key_size
   -file filename -sig_alg algorithm
  ```

  Instead of `-dn` *`distinguished_name`*, you can use `-san_dsname` *`DNS_names`*, `-san_emailaddr` *`email_addresses`*, or `-san_ipaddr` *`IP_addresses`*.
- Using **`runmqakm`**:

  ```
  runmqakm -certreq -create -db filename -pw password -label label
          -dn distinguished_name -size key_size
   -file filename -fips
          -sig_alg algorithm
  ```

where:

-db *filename*
Specifies the fully qualified file name of a CMS key database.

-pw *password*
Specifies the password for the CMS key database.

-label *label*
Specifies the key label attached to the certificate. The label is either the value of the **CERTLABL** attribute, if it is set, or the default ibmwebspheremq with the name of the queue manager or the IBM MQ MQI client logon user ID appended, all in lowercase. See "Digital certificate labels, understanding the requirements" on page 400 for details.

-dn *distinguished_name*
Specifies the X.500 distinguished name enclosed in double quotation marks. At least one attribute is required. You can supply multiple OU and DC attributes.

Note: The **runmqckm** and **runmqakm** tools refer to the postal code attribute as POSTALCODE, not PC. Always specify POSTALCODE in the **-dn** parameter when you use these certificate management commands to request certificates with a postal code.

-size *key_size*
Specifies the key size. If you are using **runmqckm**, the value can be 512 or 1024. If you are using **runmqakm**, the value can be 512, 1024, or 2048.

-file *filename*
Specifies the file name for the certificate request.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-sig_alg
For **runmqckm**, specifies the asymmetric signature algorithm used for the creation of the entry's key pair. The value can be MD2_WITH_RSA, MD2WithRSA, MD5_WITH_RSA, MD5WithRSA, SHA1WithDSA, SHA1WithRSA, SHA256_WITH_RSA, SHA256WithRSA, SHA2WithRSA, SHA384_WITH_RSA, SHA384WithRSA, SHA512_WITH_RSA, SHA512WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, SHAWithDSA, or SHAWithRSA. The default value is SHA1WithRSA

-sig_alg
For **runmqakm**, specifies the hashing algorithm used during the creation of a certificate request. This hashing algorithm is used to create the signature associated with the newly created certificate request. The value can be md5, MD5_WITH_RSA, MD5WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, sha1, SHA1WithDSA, SHA1WithECDSA, SHA1WithRSA, sha224, SHA224_WITH_RSA, SHA224WithDSA, SHA224WithECDSA, SHA224WithRSA, sha256, SHA256_WITH_RSA, SHA256WithDSA, SHA256WithECDSA, SHA256WithRSA, SHA2WithRSA, sha384, SHA384_WITH_RSA, SHA384WithECDSA, SHA384WithRSA, sha512, SHA512_WITH_RSA, SHA512WithECDSA, SHA512WithRSA, SHAWithDSA, SHAWithRSA, EC_ecdsa_with_SHA1, EC_ecdsa_with_SHA224, EC_ecdsa_with_SHA256, EC_ecdsa_with_SHA384, or EC_ecdsa_with_SHA512. The default value is SHA1WithRSA.

-san_dnsname *DNS_names*
Specifies a comma-delimited or space-delimited list of DNS names for the entry being created.

-san_emailaddr *email_addresses*
Specifies a comma-delimited or space-delimited list of email addresses for the entry being created.

-san_ipaddr *IP_addresses*
Specifies a comma-delimited or space-delimited list of IP addresses for the entry being created.

What to do next

Submit a certificate request to a CA. See "Receiving personal certificates into a key repository on UNIX, Linux and Windows systems" on page 687 for further information.

Importing a personal certificate to your PKCS #11 hardware:

Use this procedure for either a queue manager or an IBM MQ MQI client to import a personal certificate to your cryptographic hardware.

Using iKeyman:
Procedure

To request a personal certificate from the iKeyman user interface, complete the following steps:
1. Complete the steps to work with your cryptographic hardware. See "Managing certificates on PKCS #11 hardware" on page 698.
2. Click **Receive**. The Receive Certificate from a File window opens.
3. Select the **Data type** of the new personal certificate; for example, `Base64-encoded ASCII data` for a file with the `.arm` extension.
4. Type the certificate file name and location for the new personal certificate, or click **Browse** to select the name and location.
5. Click **OK**. If you already have a personal certificate in your key database a window opens, asking if you want to set the key you are adding as the default key in the database.
6. Click **Yes** or **No**. The Enter a Label window opens.
7. Enter the certificate label. The label is either the value of the **`CERTLABL`** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client logon user ID appended, all in lowercase. See Digital certificate labels for details.
8. Click **OK**. The **Personal Certificates** list shows the label of the new personal certificate you added. This label is formed by adding the cryptographic token label before the label you supplied.

Using the command line:
Procedure

To request a personal certificate from a command line, complete the following steps:
1. Open a command window that is configured for your environment.
2. Enter the appropriate command for your operating system and configuration:
   - On Windows, UNIX and Linux systems, use one of the following commands:

     ```
     runmqckm -cert -receive -file filename -crypto path
     -tokenlabel hardware_token -pw hardware_password
     -format cert_format
     ```

     ```
     runmqakm -cert -receive -file filename -crypto path
     -tokenlabel hardware_token -pw hardware_password
     -format cert_format -fips
     ```

   where:

   `-file` ***`filename`***
   Specifies the fully qualified file name of the file containing the personal certificate.

   `-crypto` ***`path`***
   Specifies the fully qualified path to the PKCS #11 library supplied with the hardware.

   `-tokenlabel` ***`hardware_token`***
   Specifies the label given to the storage part of the cryptographic hardware during installation.

-pw *hardware_password*
Specifies the password for access to the hardware.

-format *cert_format*
Specifies the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for binary DER data. The default is ASCII.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the ICC component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

Working with TLS on IBM MQ Appliance

IBM MQ Appliance has Transport Layer Security (TLS) support.

The IBM MQ Appliance has distinct commands for managing certificates. For detailed information about certificate management, see the IBM MQ Appliance documentation, TLS certificate management

Working with SSL or TLS on z/OS

z/OS

This information describes how you set up and work with the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) on z/OS.

Each topic includes examples of performing each task using RACF. You can perform similar tasks using the other external security managers.

On z/OS, you must also set the number of server subtasks that each queue manager uses for processing SSL calls, as described in "Setting the SSLTASKS parameter" on page 704.

z/OS SSL support is integral to the operating system, and is known as *System SSL*. System SSL is part of the Cryptographic Services Base element of z/OS. The Cryptographic Services Base members are installed in the *pdsname*. SIEALNKE partitioned data set (PDS). When you install System SSL, ensure that you choose the appropriate options to provide the CipherSpecs that you require.

Additional user ID requirements

z/OS

This information describes the additional requirements your user ID needs to set up and work with the TLS or SSL on z/OS.

Ensure that you have all the appropriate High Impact or Pervasive (HIPER) updates on your system.

Ensure that you have set up the following prerequisites:

- The *ssidCHIN* user ID is defined correctly in RACF, and that the *ssidCHIN* user ID has READ access to the following profiles:
  - IRR.DIGTCERT.LIST
  - IRR.DIGTCERT.LISTRING

  These variables are defined in the RACF FACILITY Class.
- The *ssidCHIN* user ID is the owner of the key ring.
- The personal certificate of the queue manager, if created by the RACDCERT command, is created with a certificate type user ID that is also the same as the *ssidCHIN* user ID.
- The channel initiator is recycled, or the command **REFRESH SECURITY TYPE(SSL)** is issued, to pick up any changes you make to the key ring.

- The IBM MQ Channel Initiator procedure has access to the system SSL runtime library *pdsname*.SIEALNKE through the link list, LPA, or a STEPLIB DD statement. This library must be APF-authorized.
- The user ID under whose authority the channel initiator is running is configured to use UNIX System Services (USS), as described in the z/OS UNIX System Services Planning documentation.

  Users who do not want the channel initiator to invoke UNIX System Services using the guest/default UID and OMVS segment, need only model a new OMVS segment based on the default segment as the channel initiator requires no special permissions, and does not run within UNIX as a superuser.

Setting the SSLTASKS parameter

Use the ALTER QMGR command to set the number of server subtasks for processing SSL calls

To use SSL channels, ensure that there are at least two server subtasks by setting the SSLTASKS parameter, using the ALTER QMGR command. For example:

```
ALTER QMGR SSLTASKS(5)
```

To avoid problems with storage allocation, do not set the SSLTASKS attribute to a value greater than eight in an environment where there is no Certificate Revocation List (CRL) checking.

If CRL checking is used, an SSLTASK is held by the channel concerned for the duration of that check. This could be for a significant elapsed time while the relevant LDAP server is contacted, because each SSLTASK is a z/OS task control block.

You must restart the channel initiator if you change the value of the SSLTASKS attribute.

Setting up a key repository on z/OS

Set up a key repository at both ends of the connection. Associate each key repository with its queue manager.

An SSL or TLS connection requires a *key repository* at each end of the connection. Each queue manager must have access to a key repository. Use the SSLKEYR parameter on the ALTER QMGR command to associate a key repository with a queue manager. See "The SSL or TLS key repository" on page 398 for more information.

On z/OS, digital certificates are stored in a *key ring* that is managed by your External Security Manager (ESM) . These digital certificates have labels, which associate the certificate with a queue manager. SSL and TLS use these certificates for authentication purposes. All the examples that follow use RACF commands. Equivalent commands exist for other ESM programs.

On z/OS, IBM MQ uses either the value of the **`CERTLABL`** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager appended. See Digital certificate labels for details.

The key repository name for a queue manager is the name of a key ring in your RACF database. You can specify the key ring name either before or after creating the key ring.

Use the following procedure to create a new key ring for a queue manager:

1. Ensure that you have the appropriate authority to issue the RACDCERT command (see the *SecureWay Security Server RACF Command Language Reference* for more details).
2. Issue the following command:

   ```
   RACDCERT ID( userid1 ) ADDRING( ring-name )
   ```

   where:

   - *userid1* is the user ID of the channel initiator address space, or the user ID that is going to own the key ring (if the key ring is shared).

- *ring-name* is the name you want to give to your key ring. The length of this name can be up to 237 characters. This name is case-sensitive. Specify *ring-name* in uppercase characters to avoid problems.

Making CA certificates available to a queue manager on z/OS:

After you have created your key ring, connect any relevant CA certificates to it.

If you have the CA certificate in a dataset, you must first add the certificate to the RACF® database by using the following command:

```
RACDCERT ID( userid1 ) ADD( input-data-set-name ) WITHLABEL( 'My CA' )
```

Then to connect a CA certificate for `My CA` to your key ring, use the following command:

```
RACDCERT ID(userid1)
CONNECT(CERTAUTH LABEL('My CA') RING(ring-name) USAGE(CERTAUTH))
```

where *userid1* is either the channel initiator user ID or the owner of a shared key ring.

For more information about CA certificates, refer to "Digital certificates" on page 382.

Locating the key repository for a queue manager on z/OS

Use this procedure to obtain the location of your queue manager's key ring.

1. Display your queue manager's attributes, using either of the following MQSC commands:
   ```
   DISPLAY QMGR ALL
   DISPLAY QMGR SSLKEYR
   ```
2. Examine the command output for the location of the key ring.

Specifying the key repository location for a queue manager

To specify the location of your queue manager's key ring, use the ALTER QMGR MQSC command to set your queue manager's key repository attribute.

For example:

```
ALTER QMGR SSLKEYR(CSQ1RING)
```

if the key ring is owned by the channel initiator address space, or:

```
ALTER QMGR SSLKEYR(userid1/CSQ1RING)
```

if it is a shared key ring, where *userid1* is the user ID that owns the key ring.

Giving the channel initiator the correct access rights

The channel initiator (CHINIT) needs access to the key repository and to certain security profiles.

Granting the CHINIT access to read the key repository

If the key repository is owned by the CHINIT user ID, this user ID needs read access to the IRR.DIGTCERT.LISTRING profile in the FACILITY class, and update access otherwise. Grant access by using the PERMIT command with ACCESS(UPDATE) or ACCESS(READ) as appropriate:

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID( userid ) ACCESS(UPDATE)
```

where *userid* is the user ID of the channel initiator address space.

Granting the CHINIT read access to the appropriate CSF* profiles

For hardware support provided through the Integrated Cryptographic Service Facility (ICSF) to be used, ensure your CHINIT user ID has read access to the appropriate CSF* profiles in the CSFSERV class by using the following command:

```
PERMIT csf-resource CLASS(CSFSERV) ID( userid ) ACCESS(READ)
```

where *csf-resource* is the name of the CSF* profile and *userid* is the user ID of the channel initiator address space.

Repeat this command for each of the following CSF* profile names:
- CSFPKD
- CSFPKE
- CSFPKI
- CSFDSG
- CSFDSV

If your certificate keys are stored in ICSF and your installation has established access control over keys stored in ICSF, ensure your CHINIT user ID has read access to the profile in the CSFKEYS class by using the following command:

```
PERMIT IRR.DIGTCERT. userid.* CLASS(CSFKEYS) ID( userid ) ACCESS(READ)
```

where *userid* is the user ID of the channel initiator address space.

Using the Integrated Cryptographic Service Facility (ICSF)

The channel initiator can use ICSF to generate a random number when seeding the password protection algorithm to obfuscate passwords flowing over client channels if SSL/TLS is not being used.

For further information, see "Using the Integrated Cryptographic Service Facility (ICSF)" on page 643

When changes to certificates or the key repository become effective on z/OS

Changes become effective when the channel initiator starts or the repository is refreshed.

Specifically, changes to the certificates in the key ring and to the key repository attribute become effective on either of the following occasions:
- When the channel initiator is started or restarted.
- When the REFRESH SECURITY TYPE(SSL) command is issued to refresh the contents of the SSL key repository.

Creating a self-signed personal certificate on z/OS

Use this procedure to create a self-signed personal certificate.

1. Generate a certificate and a public and private key pair using the following command:
   ```
   RACDCERT ID(userid2) GENCERT
   SUBJECTSDN(CN('common-name')
              T('title')
              OU('organizational-unit')
              O('organization')
              L('locality')
              SP('state-or-province')
              C('country'))
   WITHLABEL('label-name')
   ```
2. Connect the certificate to your key ring using the following command:
   ```
   RACDCERT ID(userid1)
   CONNECT(ID(userid2) LABEL('label-name') RING(ring-name) USAGE(PERSONAL))
   ```

where:
- *userid1* is the user ID of the channel initiator address space or owner of the shared key ring.
- *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space.

userid1 and *userid2* can be the same ID.

- *ring-name* is the name you gave the key ring in "Setting up a key repository on z/OS" on page 704.
- *label-name* must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager appended. See Digital certificate labels for details.

Requesting a personal certificate on z/OS

Apply for a personal certificate using RACF.

To apply for a personal certificate, use RACF as follows:

1. Create a self-signed personal certificate, as in "Creating a self-signed personal certificate on z/OS" on page 706. This certificate provides the request with the attribute values for the Distinguished Name.
2. Create a PKCS #10 Base64-encoded certificate request written to a data set, using the following command:

   ```
   RACDCERT ID(userid2) GENREQ(LABEL(' label_name ')) DSN(' output_data_set_name ')
   ```

   where
   - *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space
   - *label_name* is the label used when creating the self-signed certificate

   See "Digital certificate labels, understanding the requirements" on page 400 for details.
3. Send the data set to a Certificate Authority (CA) to request a new personal certificate.
4. When the signed certificate is returned to you by the Certificate Authority, add the certificate back into the RACF database, using the original label, as described in "Adding personal certificates to a key repository on z/OS" on page 708.

Creating a RACF signed personal certificate

RACF can function as a certificate authority and issue its own CA certificate.

This section uses the term *signer certificate* to denote a CA certificate issued by RACF.

The private key for the signer certificate must be in the RACF database before you carry out the following procedure:

1. Use the following command to generate a personal certificate signed by RACF, using the signer certificate contained in your RACF database:

   ```
   RACDCERT ID(userid2) GENCERT
   SUBJECTSDN(CN('common-name')
             T('title')
             OU('organizational-unit')
             O('organization')
             L('locality')
             SP('state-or-province')
             C('country'))
   WITHLABEL('label-name')
   SIGNWITH(CERTAUTH LABEL('signer-label'))
   ```
2. Connect the certificate to your key ring using the following command:

   ```
   RACDCERT ID(userid1)
   CONNECT(ID(userid2) LABEL('label-name') RING(ring-name) USAGE(PERSONAL))
   ```

where:

- *userid1* is the user ID of the channel initiator address space or owner of the shared key ring.
- *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space.

  userid1 and *userid2* can be the same ID.
- *ring-name* is the name you gave the key ring in "Setting up a key repository on z/OS" on page 704.

- *label-name* must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager or queue-sharing group appended. See Digital certificate labels for details.
- *signer-label* is the label of your own signer certificate.

Adding personal certificates to a key repository on z/OS

Use this procedure to add or import a personal certificate to a key ring.

After the certificate authority sends you a new personal certificate, add it to the key ring using the following procedure:

1. Add the certificate to the RACF database using the following command:

   ```
   RACDCERT ID( userid2 ) ADD( input-data-set-name ) WITHLABEL(' label-name ')
   ```

2. Connect the certificate to your key ring using the following command:

   ```
   RACDCERT ID( userid1 )
   CONNECT(ID( userid2 ) LABEL(' label-name ') RING( ring-name ) USAGE(PERSONAL))
   ```

where:

- *userid1* is the user ID of the channel initiator address space or owner of the shared key ring.
- *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space.
- *ring-name* is the name you gave the key ring in “Setting up a key repository on z/OS” on page 704.
- *input-data-set-name* is the name of the data set containing the CA signed certificate. The data set must be cataloged and must not be a PDS or a member of a PDS. The record format (RECFM) expected by RACDCERT is VB. RACDCERT dynamically allocates and opens the data set, and reads the certificate from it as binary data.
- *label-name* is the label name that was used when you created the original request. It must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager or queue-sharing group appended. See Digital certificate labels for details.

Exporting a personal certificate from a key repository on z/OS

Export the certificate using the RACDCERT command.

On the system from which you want to export the certificate, use the following command:

```
RACDCERT ID(userid2) EXPORT(LABEL('label-name'))
DSN(output-data-set-name) FORMAT(CERTB64)
```

where:

- *userid2* is the user ID under which the certificate was added to the key ring.
- *label-name* is the label of the certificate you want to extract.
- *output-data-set-name* is the data set into which the certificate is placed.
- CERTB64 is a DER encoded X.509 certificate that is in Base64 format. You can choose an alternative format, for example:

  CERTDER
  DER encoded X.509 certificate in binary format

  PKCS12B64
  PKCS #12 certificate in Base64 format

  PKCS12DER
  PKCS #12 certificate in binary format

Deleting a personal certificate from a key repository on z/OS

Delete a personal certificate using the RACDCERT command.

Before deleting a personal certificate, you might want to save a copy of it. To copy your personal certificate to a data set before deleting it, follow the procedure in "Exporting a personal certificate from a key repository on z/OS" on page 708. Then use the following command to delete your personal certificate:

```
RACDCERT ID( userid2 ) DELETE(LABEL(' label-name '))
```

where:

- *userid2* is the user ID under which the certificate was added to the key ring.
- *label-name* is the name of the certificate you want to delete.

Renaming a personal certificate in a key repository on z/OS

Rename a certificate using the RACDCERT command.

If you do not want a certificate with a specific label to be found, but do not want to delete it, you can rename it temporarily using the following command:

```
RACDCERT ID( userid2 ) LABEL(' label-name ') NEWLABEL(' new-label-name ')
```

where:

- *userid2* is the user ID under which the certificate was added to the key ring.
- *label-name* is the name of the certificate you want to rename.
- *new-label-name* is the new name of the certificate.

This can be useful when testing SSL client authentication.

Associating a user ID with a digital certificate on z/OS

IBM MQ can use a user ID associated with a RACF certificate as a channel user ID. Associate a user ID with a certificate by installing it under that user ID, or using a Certificate Name Filter.

The method described in this topic is an alternative to the platform-independent method for associating a user ID with a digital certificate, which uses channel authentication records. For more information about channel authentication records, see "Channel authentication records" on page 424.

When an entity at one end of an SSL channel receives a certificate from a remote connection, the entity asks RACF if there is a user ID associated with that certificate. The entity uses that user ID as the channel user ID. If there is no user ID associated with the certificate, the entity uses the user ID under which the channel initiator is running.

Associate a user ID with a certificate in either of the following ways:

- Install that certificate into the RACF database under the user ID with which you want to associate it, as described in "Adding personal certificates to a key repository on z/OS" on page 708.
- Use a Certificate Name Filter (CNF) to map the Distinguished Name of the subject or issuer of the certificate to the user ID, as described in "Setting up a certificate name filter" on page 710.

Setting up a certificate name filter:

Use the RACDCERT command to define a certificate name filter (CNF), which maps a Distinguished Name to a user ID.

Perform the following steps to set up a CNF.

1. Enable CNF functions using the following command. You require update authority on the class DIGTNMAP to do this.
   ```
   SETROPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
   ```
2. Define the CNF. For example:
   ```
   RACDCERT ID(USER1) MAP WITHLABEL('filter1') TRUST
   SDNFILTER('O=IBM.C=UK') IDNFILTER('O=ExampleCA.L=Internet')
   ```
   where `USER1` is the user ID to be used when:
   - The DN of the subject has an Organization of `IBM` and a Country of `UK`.
   - The DN of the issuer has an Organization of `ExampleCA` and a Locality of `Internet`.
3. Refresh the CNF mappings:
   ```
   SETROPTS RACLIST(DIGTNMAP) REFRESH
   ```

Note:

1. If the actual certificate is stored in the RACF database, the user ID under which it is installed is used in preference to the user ID associated with any CNF. If the certificate is not stored in the RACF database, the user ID associated with the most specific matching CNF is used. Matches of the subject DN are considered more specific than matches of the issuer DN.
2. Changes to CNFs do not apply until you refresh the CNF mappings.
3. A DN matches the DN filter in a CNF only if the DN filter is identical to the *least significant portion* of the DN. The least significant portion of the DN comprises the attributes that are usually listed at the right-most end of the DN, but which appear at the beginning of the certificate.

   For example, consider the SDNFILTER `'O=IBM.C=UK'`. A subject DN of `'CN=QM1.O=IBM.C=UK'` matches that filter, but a subject DN of `'CN=QM1.O=IBM.L=Hursley.C=UK'` does not match that filter.

   The least significant portion of some certificates can contain fields that do not match the DN filter. Consider excluding these certificates by specifying a DN pattern in the SSLPEER pattern on the DEFINE CHANNEL command.
4. If the most specific matching CNF is defined to RACF as NOTRUST, the entity uses the user ID under which the channel initiator is running.
5. RACF uses the `'.'` character as a separator. IBM MQ uses either a comma or a semicolon.

You can define CNFs to ensure that the entity never sets the channel user ID to the default, which is the user ID under which the channel initiator is running. For each CA certificate in the key ring associated with the entity, define a CNF with an IDNFILTER that exactly matches the subject DN of that CA certificate. This ensures that all certificates that the entity might use match at least one of these CNFs. This is because all such certificates must either be connected to the key ring associated with the entity, or must be issued by a CA for which a certificate is connected to the key ring associated with the entity.

Refer to the *SecureWay Security Server RACF Security Administrator's Guide* for more information about the commands you use to manipulate CNFs.

Defining a sender channel and transmission queue on QMA

Use the **DEFINE CHANNEL** and **DEFINE QLOCAL** commands to set up the required objects.

Procedure

On QMA, issue commands like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QMB.MACH.COM) XMITQ(QMB)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA) DESCR('Sender channel using TLS from QMA to QMB')

DEFINE QLOCAL(QMB) USAGE(XMITQ)
```

Results

A sender channel, TO.QMB, and a transmission queue, QMB, are created.

Defining a receiver channel on QMB

Use the **DEFINE CHANNEL** command to set up the required object.

Procedure

On QMB, issue a command like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA)
SSLCAUTH(REQUIRED) DESCR('Receiver channel using TLS to QMB')
```

Results

A receiver channel, TO.QMB, is created.

Starting the sender channel

If necessary, start a listener program and refresh security. Then start the channel using the **START CHANNEL** command.

Procedure

1. Optional: If you have not already done so, start a listener program on QMB. The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener.
2. Optional: If any SSL or TLS channels have run previously, issue the command `REFRESH SECURITY TYPE(SSL)`. This ensures that all the changes made to the key repository are available.
3. Start the channel on QMA, using the command `START CHANNEL(TO.QMB)`.

Results

The sender channel is started.

Exchanging self-signed certificates

Exchange the certificates you previously extracted. If you use FTP, use the correct format.

Procedure

Transfer the CA part of the QM1 certificate to the QM2 system and vice versa, for example, by FTP.
If you transfer the certificates using FTP, you must do so in the correct format.
Transfer the following certificate types in *binary* format:

- DER encoded binary X.509
- PKCS #7 (CA certificates)
- PKCS #12 (personal certificates)

Transfer the following certificate types in ASCII format:

- PEM (privacy-enhanced mail)
- Base64 encoded X.509

Defining a sender channel and transmission queue on QM1

Use the **DEFINE CHANNEL** and **DEFINE QLOCAL** commands to set up the required objects.

Procedure

On QM1, issue commands like the following example:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM1.MACH.COM) XMITQ(QM2) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA)
```

```
DEFINE QLOCAL(QM2) USAGE(XMITQ)
```

This example uses CipherSpec RC4_MD5. The CipherSpecs at each end of the channel must be the same. Only the SSLCIPH parameter is mandatory if you want your channel to use SSL or TLS. See "CipherSpecs and CipherSuites in IBM MQ" on page 413 for information about the permitted values for the SSLCIPH parameter.

Results

A sender channel, QM1.TO.QM2, and a transmission queue, QM2, are created.

Defining a receiver channel on QM2

Use the **DEFINE CHANNEL** command to set up the required object.

Procedure

On QM2, issue a command like the following example:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA)
SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL from QM1 to QM2')
```

The channel must have the same name as the sender channel you defined in "Defining a sender channel and transmission queue on QM1," and use the same CipherSpec.

Starting the sender channel

If necessary, start a listener program and refresh security. Then start the channel using the **START CHANNEL** command.

Procedure

1. Optional: If you have not already done so, start a listener program on QM2. The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener
2. Optional: If any SSL or TLS channels have run previously, issue the command REFRESH SECURITY TYPE(SSL). This ensures that all the changes made to the key repository are available.
3. On QM1, start the channel, using the command `START CHANNEL(QM1.TO.QM2)`.

Results

The sender channel is started.

Refreshing the SSL or TLS environment

Refresh the SSL or TLS environment on queue manager QMA using the **REFRESH SECURITY** command.

Procedure

On QMA, enter the following command:

```
REFRESH SECURITY TYPE(SSL)
```

This ensures that all the changes made to the key repository are available.

Allowing anonymous connections on a receiver channel

Use the **ALTER CHANNEL** command to make SSL or TLS client authentication optional.

Procedure

On QMB, enter the following command:

```
ALTER CHANNEL(TO.QMB) CHLTYPE(RCVR) SSLCAUTH(OPTIONAL)
```

Starting the sender channel

If necessary, start the channel initiator, start a listener program, and refresh security. Then start the channel using the **START CHANNEL** command.

Procedure

1. Optional: if you have not already done so, start the channel initiator.
2. Optional: If you have not already done so, start a listener program on QM2. The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener
3. Optional: If the channel initiator was already running or any SSL or TLS channels have run previously, issue the command REFRESH SECURITY TYPE(SSL). This ensures that all the changes made to the key repository are available.
4. On QM1, start the channel, using the command `START CHANNEL(QM1.TO.QM2)`.

Results

The sender channel is started.

Starting the sender channel on z/OS

If necessary, start the channel initiator, start a listener program, and refresh security. Then start the channel using the **START CHANNEL** command.

Procedure

1. Optional: If you have not already done so, start the channel initiator.
2. Optional: If you have not already done so, start a listener program on QMB. The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener.
3. Optional: If the channel initiator was already running or if any SSL or TLS channels have run previously, issue the command `REFRESH SECURITY TYPE(SSL)`. This ensures that all the changes made to the key repository are available.
4. Start the channel on QMA, using the command `START CHANNEL(TO.QMB)`.

Results

The sender channel is started.

Identifying and authenticating users

You can identify and authenticate users by using the MQCSP structure or in several types of user exit program.

Using the MQCSP structure

You specify the MQCSP connection security parameters structure on an MQCONNX call; this structure contains a user ID and password. If necessary, you can alter the MQCSP in a security exit.

Note: The object authority manager (OAM) does not use the password. However the OAM does some limited work with the user ID, that could be considered a trivial form of authentication. These checks stop you adopting another user ID, if you use those parameters in your applications.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see "MQCSP password protection" on page 404.

Implementing identification and authentication in security exits

The primary purpose of a security exit is to enable the MCA at each end of a channel to authenticate its partner. At each end of a message channel, and at the server end of an MQI channel, an MCA typically acts on behalf of the queue manager to which it is connected. At the client end of an MQI channel, an MCA typically acts on behalf of the user of the IBM MQ client application. In this situation, mutual authentication actually takes place between two queue managers, or between a queue manager and the user of an IBM MQ MQI client application.

The supplied security exit (the SSPI channel exit) illustrates how mutual authentication can be implemented by exchanging authentication tokens that are generated, and then checked, by a trusted authentication server such as Kerberos. For more details, see "The SSPI channel exit program" on page 533.

Mutual authentication can also be implemented by using Public Key Infrastructure (PKI) technology. Each security exit generates some random data, signs it using the private key of the queue manager or user it is representing, and sends the signed data to its partner in a security message. The partner security exit performs the authentication by checking the digital signature using the public key of the queue manager or user. Before exchanging digital signatures, the security exits might need to agree the algorithm for generating a message digest, if more than one algorithm is available for use.

When a security exit sends the signed data to its partner, it also needs to send some means of identifying the queue manager or user it is representing. This might be a Distinguished Name, or even a digital certificate. If a digital certificate is sent, the partner security exit can validate the certificate by working through the certificate chain to the root CA certificate. This provides assurance of the ownership of the public key that is used to check the digital signature.

The partner security exit can validate a digital certificate only if it has access to a key repository that contains the remaining certificates in the certificate chain. If a digital certificate for the queue manager or user is not sent, one must be available in the key repository to which the partner security exit has access. The partner security exit cannot check the digital signature unless it can find the signer's public key.

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) use PKI techniques like the ones just described. For more information about how SSL and TLS perform authentication, see "Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts" on page 388.

If a trusted authentication server or PKI support is not available, other techniques can be used. A common technique, which can be implemented in security exits, uses a symmetric key algorithm.

One of the security exits, exit A, generates a random number and sends it in a security message to its partner security exit, exit B. Exit B encrypts the number using its copy of a key which is known only to the two security exits. Exit B sends the encrypted number to exit A in a security message with a second random number that exit B has generated. Exit A verifies that the first random number has been encrypted correctly, encrypts the second random number using its copy of the key, and sends the encrypted number to exit B in a security message. Exit B then verifies that the second random number has been encrypted correctly. During this exchange, if either security exit is not satisfied with the authenticity of other, it can instruct the MCA to close the channel.

An advantage of this technique is that no key or password is sent over the communications connection during the exchange. A disadvantage is that it does not provide a solution to the problem of how to distribute the shared key in a secure way. One solution to this problem is described in "Implementing confidentiality in user exit programs" on page 806. A similar technique is used in SNA for the mutual authentication of two LUs when they bind to form a session. The technique is described in "Session level authentication" on page 493.

All the preceding techniques for mutual authentication can be adapted to provide one-way authentication.

Implementing identification and authentication in message exits

When an application puts a message on a queue, the *UserIdentifier* field in the message descriptor contains a user ID associated with the application. However, there is no data present that can be used to authenticate the user ID. This data can be added by a message exit at the sending end of a channel and checked by a message exit at the receiving end of the channel. The authenticating data can be an encrypted password or a digital signature, for example.

This service might be more effective if it is implemented at the application level. The basic requirement is for the user of the application that receives the message to be able to identify and authenticate the user of the application that sent the message. It is therefore natural to consider implementing this service at the application level. For more information, see "Identity mapping in the API exit and API-crossing exit" on page 720.

Implementing identification and authentication in the API exit and API-crossing exit

At the level of an individual message, identification and authentication is a service that involves two users, the sender and the receiver of the message. The basic requirement is for the user of the application that receives the message to be able to identify and authenticate the user of the application that sent the message. Note that the requirement is for one way, not two way, authentication.

Depending on how it is implemented, the users and their applications might need to interface, or even interact, with the service. In addition, when and how the service is used might depend on where the users and their applications are located, and on the nature of the applications themselves. It is therefore natural to consider implementing the service at the application level rather than at the link level.

If you consider implementing this service at the link level, you might need to resolve issues such as the following:

- On a message channel, how do you apply the service only to those messages that require it?
- How do you enable users and their applications to interface, or interact, with the service, if this is a requirement?

- In a multi-hop situation, where a message is sent over more than one message channel on the way to its destination, where do you invoke the components of the service?

Here are some examples of how the identification and authentication service can be implemented at the application level. The term *API exit* means either an API exit or an API-crossing exit.

- When an application puts a message on a queue, an API exit can acquire an authentication token from a trusted authentication server such as Kerberos. The API exit can add this token to the application data in the message. When the message is retrieved by the receiving application, a second API exit can ask the authentication server to authenticate the sender by checking the token.
- When an application puts a message on a queue, an API exit can append the following items to the application data in the message:
  - The digital certificate of the sender
  - The digital signature of the sender

  If different algorithms for generating a message digest are available for use, the API exit can include the name of the algorithm it has used.

  When the message is retrieved by the receiving application, a second API exit can perform the following checks:
  - The API exit can validate the digital certificate by working through the certificate chain to the root CA certificate. To do this, the API exit must have access to a key repository that contains the remaining certificates in the certificate chain. This check provide assurance that the sender, identified by the Distinguished Name, is the genuine owner of the public key contained in the certificate.
  - The API exit can check the digital signature using the public key contained in the certificate. This check authenticates the sender.

  The Distinguished Name of the sender can be sent instead of the whole digital certificate. In this case, the key repository must contain the sender's certificate so that the second API exit can find the public key of the sender. Another possibility is to send all the certificates in the certificate chain.
- When an application puts a message on a queue, the *UserIdentifier* field in the message descriptor contains a user ID associated with the application. The user ID can be used to identify the sender. To enable authentication, an API exit can append some data, such as an encrypted password, to the application data in the message. When the message is retrieved by the receiving application, a second API exit can authenticate the user ID by using the data that has travelled with the message.

  This technique might be considered sufficient for messages that originate in a controlled and trusted environment, and in circumstances where a trusted authentication server or PKI support is not available.

UNIX Linux V 8.0.0.3

Pluggable Authentication Method (PAM)

PAM is now common across UNIX and Linux platforms, and provides a general mechanism that hides the details of user authentication from services.

Different authentication rules can be used for different services, by configuring the rules, without any change needed to the services themselves.

See "Using the Pluggable Authentication Method (PAM)" on page 732 for further information.

Privileged users

A privileged user is one that has full administrative authorities for IBM MQ.

In addition to the users listed in the following table, members of any group with `+connect` authority for queue managers, and `+crt` authority for queues are indirectly administrators. Similarly, any user that has `+connect` and `+set` authority on the queue manager, and `+put` authority on the command queue is an administrator.

You should not grant these privileges to ordinary users and applications.

Table 78. Privileged users by platform

Platform	Privileged users
Windows systems	• SYSTEM • Members of the mqm group • Members of the Administrators group
UNIX and Linux systems	• Members of the mqm group
IBM i IBM i systems	• The profiles qmqm and qmqmadm • All members of the qmqmadm group • Any user defined with the *ALLOBJ setting
z/OS	The user ID that the channel initiator, queue manager and advanced message security address spaces are running under.

Identifying and authenticating users using the MQCSP structure

You can specify the MQCSP connection security parameters structure on an MQCONNX call.

The MQCSP connection security parameters structure contains a user ID and password, which the authorization service can use to identify and authenticate the user.

The authorization service component supplied with IBM MQ is called the Object Authority Manager (OAM). The OAM authorizes users based on the ID contained in the MQCSP but does not validate the password. It is possible to implement password validation in the authorization service by using chained exits with the OAM, or by replacing the OAM with an alternative authorization service.

You can alter the MQCSP in a security exit.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see "MQCSP password protection" on page 404.

Implementing identification and authentication in security exits

You can use a security exit to implement one-way or mutual authentication.

The primary purpose of a security exit is to enable the MCA at each end of a channel to authenticate its partner. At each end of a message channel, and at the server end of an MQI channel, an MCA typically acts on behalf of the queue manager to which it is connected. At the client end of an MQI channel, an MCA typically acts on behalf of the user of the IBM MQ MQI client application. In this situation, mutual authentication actually takes place between two queue managers, or between a queue manager and the user of an IBM MQ MQI client application.

The supplied security exit (the SSPI channel exit) illustrates how mutual authentication can be implemented by exchanging authentication tokens that are generated, and then checked, by a trusted authentication server such as Kerberos. For more details, see "The SSPI channel exit program" on page 533.

Mutual authentication can also be implemented by using Public Key Infrastructure (PKI) technology. Each security exit generates some random data, signs it using the private key of the queue manager or user it is representing, and sends the signed data to its partner in a security message. The partner security exit performs the authentication by checking the digital signature using the public key of the queue manager or user. Before exchanging digital signatures, the security exits might need to agree the algorithm for generating a message digest, if more than one algorithm is available for use.

When a security exit sends the signed data to its partner, it also needs to send some means of identifying the queue manager or user it is representing. This might be a Distinguished Name, or even a digital certificate. If a digital certificate is sent, the partner security exit can validate the certificate by working through the certificate chain to the root CA certificate. This provides assurance of the ownership of the public key that is used to check the digital signature.

The partner security exit can validate a digital certificate only if it has access to a key repository that contains the remaining certificates in the certificate chain. If a digital certificate for the queue manager or user is not sent, one must be available in the key repository to which the partner security exit has access. The partner security exit cannot check the digital signature unless it can find the signer's public key.

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) use PKI techniques like the ones just described. For more information about how the Secure Sockets Layer performs authentication, see "Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts" on page 388.

If a trusted authentication server or PKI support is not available, other techniques can be used. A common technique, which can be implemented in security exits, uses a symmetric key algorithm.

One of the security exits, exit A, generates a random number and sends it in a security message to its partner security exit, exit B. Exit B encrypts the number using its copy of a key which is known only to the two security exits. Exit B sends the encrypted number to exit A in a security message with a second random number that exit B has generated. Exit A verifies that the first random number has been encrypted correctly, encrypts the second random number using its copy of the key, and sends the encrypted number to exit B in a security message. Exit B then verifies that the second random number has been encrypted correctly. During this exchange, if either security exit is not satisfied with the authenticity of other, it can instruct the MCA to close the channel.

An advantage of this technique is that no key or password is sent over the communications connection during the exchange. A disadvantage is that it does not provide a solution to the problem of how to distribute the shared key in a secure way. One solution to this problem is described in "Implementing confidentiality in user exit programs" on page 806. A similar technique is used in SNA for the mutual authentication of two LUs when they bind to form a session. The technique is described in "Session level authentication" on page 493.

All the preceding techniques for mutual authentication can be adapted to provide one-way authentication.

Identity mapping in message exits

You can use message exits to process information to authenticate a user ID, though it might be better to implement authentication at the application level.

When an application puts a message on a queue, the *UserIdentifier* field in the message descriptor contains a user ID associated with the application. However, there is no data present that can be used to authenticate the user ID. This data can be added by a message exit at the sending end of a channel and checked by a message exit at the receiving end of the channel. The authenticating data can be an encrypted password or a digital signature, for example.

This service might be more effective if it is implemented at the application level. The basic requirement is for the user of the application that receives the message to be able to identify and authenticate the user of the application that sent the message. It is therefore natural to consider implementing this service at the application level. For more information, see "Identity mapping in the API exit and API-crossing exit."

Identity mapping in the API exit and API-crossing exit

An application that receives a message must be able to identify and authenticate the user of the application that sent the message. This service is typically best implemented at the application level. API exits can implement the service in a number of ways.

At the level of an individual message, identification and authentication is a service that involves two users, the sender and the receiver of the message. The basic requirement is for the user of the application that receives the message to be able to identify and authenticate the user of the application that sent the message. Note that the requirement is for one way, not two way, authentication.

Depending on how it is implemented, the users and their applications might need to interface, or even interact, with the service. In addition, when and how the service is used might depend on where the users and their applications are located, and on the nature of the applications themselves. It is therefore natural to consider implementing the service at the application level rather than at the link level.

If you consider implementing this service at the link level, you might need to resolve issues such as the following:

- On a message channel, how do you apply the service only to those messages that require it?
- How do you enable users and their applications to interface, or interact, with the service, if this is a requirement?
- In a multi-hop situation, where a message is sent over more than one message channel on the way to its destination, where do you invoke the components of the service?

Here are some examples of how the identification and authentication service can be implemented at the application level. The term *API exit* means either an API exit or an API-crossing exit.

- When an application puts a message on a queue, an API exit can acquire an authentication token from a trusted authentication server such as Kerberos. The API exit can add this token to the application data in the message. When the message is retrieved by the receiving application, a second API exit can ask the authentication server to authenticate the sender by checking the token.
- When an application puts a message on a queue, an API exit can append the following items to the application data in the message:
  - The digital certificate of the sender
  - The digital signature of the sender

If different algorithms for generating a message digest are available for use, the API exit can include the name of the algorithm it has used.

When the message is retrieved by the receiving application, a second API exit can perform the following checks:

- The API exit can validate the digital certificate by working through the certificate chain to the root CA certificate. To do this, the API exit must have access to a key repository that contains the remaining certificates in the certificate chain. This check provide assurance that the sender, identified by the Distinguished Name, is the genuine owner of the public key contained in the certificate.
- The API exit can check the digital signature using the public key contained in the certificate. This check authenticates the sender.

The Distinguished Name of the sender can be sent instead of the whole digital certificate. In this case, the key repository must contain the sender's certificate so that the second API exit can find the public key of the sender. Another possibility is to send all the certificates in the certificate chain.

- When an application puts a message on a queue, the *UserIdentifier* field in the message descriptor contains a user ID associated with the application. The user ID can be used to identify the sender. To enable authentication, an API exit can append some data, such as an encrypted password, to the application data in the message. When the message is retrieved by the receiving application, a second API exit can authenticate the user ID by using the data that has travelled with the message.

  This technique might be considered sufficient for messages that originate in a controlled and trusted environment, and in circumstances where a trusted authentication server or PKI support is not available.

Working with revoked certificates

Digital certificates can be revoked by Certificate Authorities. You can check the revocation status of certificates using OCSP, or CRLs on LDAP servers, depending on platform.

During the SSL handshake, the communicating partners authenticate each other with digital certificates. Authentication can include a check that the certificate received can still be trusted. Certificate Authorities (CAs) revoke certificates for various reasons, including:

- The owner has moved to a different organization
- The private key is no longer secret

CAs publish revoked personal certificates in a Certificate Revocation List (CRL). CA certificates that have been revoked are published in an Authority Revocation List (ARL).

On the following platforms, IBM MQ SSL support checks for revoked certificates using OCSP (Online Certificate Status Protocol) or using CRLs and ARLs on LDAP (Lightweight Directory Access Protocol) servers. OCSP is the preferred method.

- Linux Linux
- UNIX UNIX
- Windows Windows

IBM MQ classes for Java and IBM MQ classes for JMS cannot use the OCSP information in a client channel definition table file. However, you can configure OCSP as described in Using Online Certificate Protocol.

On the following platforms, and IBM MQ SSL support checks for revoked certificates using CRLs and ARLs on LDAP servers only:

- IBM i IBM i
- z/OS z/OS

For more information about Certificate Authorities, see “Digital certificates” on page 382.

Revoked certificates and OCSP

IBM MQ determines which Online Certificate Status Protocol (OCSP) responder to use, and handles the response received. You might have to take steps to make the OCSP responder accessible.

Note: This information applies only to IBM MQ on Windows, UNIX and Linux systems.

To check the revocation status of a digital certificate using OCSP, IBM MQ can use two methods to determines which OCSP responder to contact:
- By using the AuthorityInfoAccess (AIA) certificate extension in the certificate to be checked.
- By using a URL specified in an authentication information object or specified by a client application.

A URL specified in an authentication information object or by a client application takes priority over a URL in an AIA certificate extension.

If the URL of the OCSP responder lies behind a firewall, reconfigure the firewall so the OCSP responder can be accessed or set up an OCSP proxy server. Specify the name of the proxy server by using the SSLHTTPProxyName variable in the SSL stanza. On client systems, you can also specify the name of the proxy server by using the environment variable MQSSLPROXY. For more details, see the related information.

If you are not concerned whether TLS or SSL certificates are revoked, perhaps because you are running in a test environment, you can set OCSPCheckExtensions to NO in the SSL stanza. If you set this variable, any AIA certificate extension is ignored. This solution is unlikely to be acceptable in a production environment, where you probably do not want to allow access from users presenting revoked certificates.

The call to access the OCSP responder can result in one of the following three outcomes:

Good The certificate is valid.

Revoked
The certificate is revoked.

Unknown
This outcome can arise for one of three reasons:
- IBM MQ cannot access the OCSP responder.
- The OCSP responder has sent a response, but IBM MQ cannot verify the digital signature of the response.
- The OCSP responder has sent a response that indicates that it has no revocation data for the certificate.

If IBM MQ receives an OCSP outcome of `Unknown`, its behavior depends on the setting of the OCSPAuthentication attribute. For queue managers, this attribute is held in the SSL stanza of the `qm.ini` file for UNIX and Linux systems, or the Windows registry. It can be set using the IBM MQ Explorer. For clients, it is held in the SSL stanza of the client configuration file.

If an outcome of `Unknown` is received and OCSPAuthentication is set to REQUIRED (the default value), IBM MQ rejects the connection and issues an error message of type AMQ9716. If queue manager SSL event messages are enabled, an SSL event message of type MQRC_CHANNEL_SSL_ERROR with ReasonQualifier set to MQRQ_SSL_HANDSHAKE_ERROR is generated.

If an outcome of `Unknown` is received and OCSPAuthentication is set to OPTIONAL, IBM MQ allows the SSL channel to start and no warnings or SSL event messages are generated.

If an outcome of `Unknown` is received and OCSPAuthentication is set to WARN, the SSL channel starts but IBM MQ issues a warning message of type AMQ9717 in the error log. If queue manager SSL event messages are enabled, an SSL event message of type

MQRC_CHANNEL_SSL_WARNING with ReasonQualifier set to MQRQ_SSL_UNKNOWN_REVOCATION is generated.

Digital signing of OCSP responses

An OCSP responder can sign its responses in one of three ways. Your responder will inform you which method is used.

- The OCSP response can be digitally signed using the same CA certificate that issued the certificate that you are checking. In this case, you do not need to set up any additional certificate; the steps you have already taken to establish SSL connectivity are sufficient to verify the OCSP response.
- The OCSP response can be digitally signed using another certificate signed by the same certificate authority (CA) that issued the certificate you are checking. The signing certificate is sent together with the OCSP response in this case. The certificate flowed from the OCSP responder must have an Extended Key Usage Extension set to `id-kp-OCSPSigning` so that it can be trusted for this purpose. Because the OCSP response is sent with the certificate which signed it (and that certificate is signed by a CA that is already trusted for SSL connectivity), no additional certificate setup is required.
- The OCSP response can be digitally signed using another certificate that is not directly related to the certificate you are checking. In this case, the OCSP response is signed by a certificate issued by the OCSP responder itself. You must add a copy of the OCSP responder certificate to the key database of the client or queue manager which performs the OCSP checking . When a CA certificate is added, by default it is added as a trusted root, which is the required setting in this context. If this certificate is not added, IBM MQ cannot verify the digital signature on the OCSP response and the OCSP check results in an Unknown outcome, which might cause IBM MQ to close the channel, depending on the value of OCSPAuthentication.

Online Certificate Status Protocol (OCSP) in Java and JMS client applications

Due to a limitation of the Java API, IBM MQ can use Online Certificate Status Protocol (OCSP) certificate revocation checking for SSL and TLS secure sockets only when OCSP is enabled for the entire Java virtual machine (JVM) process. There are two ways to enable OCSP for all secure sockets in the JVM:

- Edit the JRE java.security file to include the OCSP configuration settings that are shown in Table 1 and restart the application.
- Use the java.security.Security.setProperty() API, subject to any Java Security Manager policy in effect.

As a minimum, you must specify one of the ocsp.enable and ocsp.responderURL values.

Property Name	Description
ocsp.enable	This property's value is either `true` or `false`. If `true`, OCSP checking is enabled when doing certificate revocation checking; if `false` or not set, OCSP checking is disabled.
ocsp.responderURL	This property's value is a URL that identifies the location of the OCSP responder. Here is an example; `ocsp.responderURL=http://ocsp.example.net:80`. By default, the location of the OCSP responder is determined implicitly from the certificate that is being validated. The property is used when the Authority Information Access extension (defined in RFC 3280) is absent from the certificate or when it requires overriding.

Property Name	Description
ocsp.responderCertSubjectName	This property's value is the subject name of the OCSP responder's certificate. Here is an example; `ocsp.responderCertSubjectName="CN=OCSP Responder, O=XYZ Corp"`. By default, the certificate of the OCSP responder is that of the issuer of the certificate that is being validated. This property identifies the certificate of the OCSP responder when the default does not apply. Its value is a string distinguished name (defined in RFC 2253) which identifies a certificate in the set of certificates that are supplied during cert path validation. In cases where the subject name alone is not sufficient to uniquely identify the certificate, then both the ocsp.responderCertIssuerName and ocsp.responderCertSerialNumber properties must be used instead. When this property is set, then the properties ocsp.responderCertIssuerName and ocsp.responderCertSerialNumber are ignored.
ocsp.responderCertIssuerName	This property's value is the issuer name of the OCSP responder's certificate. Here is an example; `ocsp.responderCertIssuerName="CN=Enterprise CA, O=XYZ Corp"`. By default, the certificate of the OCSP responder is that of the issuer of the certificate that is being validated. This property identifies the certificate of the OCSP responder when the default does not apply. Its value is a string distinguished name (defined in RFC 2253) which identifies a certificate in the set of certificates that are supplied during cert path validation. When this property is set then the ocsp.responderCertSerialNumber property must also be set. This property is ignored when the ocsp.responderCertSubjectName property is set.
ocsp.responderCertSerialNumber	This property's value is the serial number of the OCSP responder's certificate. Here is an example; `ocsp.responderCertSerialNumber=2A:FF:00`. By default, the certificate of the OCSP responder is that of the issuer of the certificate that is being validated. This property identifies the certificate of the OCSP responder when the default does not apply. This value is a string of hexadecimal digits (colon or space separators might be present) which identifies a certificate in the set of certificates that are supplied during cert path validation. When this property is set then the ocsp.responderCertIssuerName property must also be set. This property is ignored when the ocsp.responderCertSubjectName property is set.

Before you enable OCSP in this way, there are a number of considerations:

- Setting the OCSP configuration affects all secure sockets in the JVM process. In some cases this configuration might have undesirable side-effects when the JVM is shared with other application code that uses SSL or TLS secure sockets. Ensure that the chosen OCSP configuration is suitable for all of the applications that are running in the same JVM.
- Applying maintenance to your JRE might overwrite the java.security file. Take care when you apply Java interim fixes and product maintenance to avoid overwriting the java.security file. It might be necessary to reapply your java.security changes after you apply maintenance. For this reason, you might consider setting the OCSP configuration by using the java.security.Security.setProperty() API instead.

- Enabling OCSP checking has an effect only if revocation checking is also enabled. Revocation checking is enabled by the `PKIXParameters.setRevocationEnabled()` method.
- If you are using the AMS Java Interceptor described in Enabling OCSP checking in native interceptors, take care to avoid using a java.security OCSP configuration that conflicts with the AMS OCSP configuration in the keystore configuration file.

Working with Certificate Revocation Lists and Authority Revocation Lists

IBM MQ support for CRLs and ARLs varies by platform.

CRL and ARL support on each platform is as follows:
- On z/OS, System SSL supports CRLs and ARLs stored in LDAP servers by the Tivoli Public Key Infrastructure product.
- On other platforms, the CRL and ARL support complies with PKIX X.509 V2 CRL profile recommendations.

IBM MQ maintains a cache of CRLs and ARLs that have been accessed in the preceding 12 hours.

When a queue manager or IBM MQ MQI client receives a certificate, it checks the CRL to confirm that the certificate is still valid. IBM MQ first checks in the cache, if there is a cache. If the CRL is not in the cache, IBM MQ interrogates the LDAP CRL server locations in the order they occur in the namelist of authentication information objects specified by the *SSLCRLNamelist* attribute, until IBM MQ finds an available CRL. If the namelist is not specified, or is specified with a blank value, CRLs are not checked.

Setting up LDAP servers

Configure the LDAP Directory Information Tree structure to reflect the hierarchy of Distinguished Names of CAs. Do this using LDAP Data Interchange Format files.

Configure the LDAP Directory Information Tree (DIT) structure to use the hierarchy corresponding to the Distinguished Names of the CAs that issue the certificates and CRLs. You can set up the DIT structure with a file that uses the LDAP Data Interchange Format (LDIF). You can also use LDIF files to update a directory.

LDIF files are ASCII text files that contain the information required to define objects within an LDAP directory. LDIF files contain one or more entries, each of which comprises a Distinguished Name, at least one object class definition and, optionally, multiple attribute definitions.

The `certificateRevocationList;binary` attribute contains a list, in binary form, of revoked user certificates. The `authorityRevocationList;binary` attribute contains a binary list of CA certificates that have been revoked. For use with IBM MQ SSL, the binary data for these attributes must conform to DER (Definite Encoding Rules) format. For more information about LDIF files, refer to the documentation provided with your LDAP server.

Figure 73 on page 726 shows a sample LDIF file that you might create as input to your LDAP server to load the CRLs and ARLs issued by CA1, which is an imaginary Certificate Authority with the Distinguished Name “CN=CA1, OU=Test, O=IBM, C=GB”, set up by the Test organization within IBM.

```
dn: o=IBM, c=GB
o: IBM
objectclass: top
objectclass: organization

dn: ou=Test, o=IBM, c=GB
ou: Test
objectclass: organizationalUnit

dn: cn=CA1, ou=Test, o=IBM, c=GB
cn: CA1
objectclass: cRLDistributionPoint
objectclass: certificateAuthority
authorityRevocationList;binary:: (DER format data)
certificateRevocationList;binary:: (DER format data)
caCertificate;binary:: (DER format data)
```

Figure 73. Sample LDIF file for a Certificate Authority. This might vary from implementation to implementation.

Figure 74 shows the DIT structure that your LDAP server creates when you load the sample LDIF file shown in Figure 73 together with a similar file for CA2, an imaginary Certificate Authority set up by the PKI organization, also within IBM.

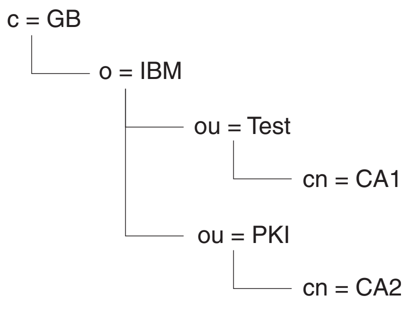


Figure 74. Example of an LDAP Directory Information Tree structure

WebSphere MQ checks both CRLs and ARLs.

Note: Ensure that the access control list for your LDAP server allows authorized users to read, search, and compare the entries that hold the CRLs and ARLs. WebSphere MQ accesses the LDAP server using the LDAPUSER and LDAPPWD properties of the AUTHINFO object.

Configuring and updating LDAP servers:

Use this procedure to configure or update your LDAP server.

1. Obtain the CRLs and ARLs in DER format from your Certification Authority, or Authorities.
2. Using a text editor or the tool provided with your LDAP server, create one or more LDIF files that contain the Distinguished Name of the CA and the required object class definitions. Copy the DER format data into the LDIF file as the values of either the `certificateRevocationList;binary` attribute for CRLs, the `authorityRevocationList;binary` attribute for ARLs , or both.
3. Start your LDAP server.
4. Add the entries from the LDIF file or files you created at step 2.

After you have configured your LDAP CRL server, check that it is set up correctly. First, try using a certificate that is not revoked on the channel, and check that the channel starts correctly. Then use a certificate that is revoked, and check that the channel fails to start.

Obtain updated CRLs from the Certification Authorities frequently. Consider doing this on your LDAP servers every 12 hours.

Accessing CRLs and ARLs with a queue manager

A queue manager is associated with one or more authentication information objects, which hold the address of an LDAP CRL server. IBM i Websphere MQ on IBM i behaves differently from other platforms.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

You tell the queue manager how to access CRLs by supplying the queue manager with authentication information objects, each of which holds the address of an LDAP CRL server. The authentication information objects are held in a namelist, which is specified in the *SSLCRLNamelist* queue manager attribute.

In the following example, MQSC is used to specify the parameters:

1. Define authentication information objects using the DEFINE AUTHINFO MQSC command, with the AUTHTYPE parameter set to CRLLDAP. IBM i On IBM i, you can also use the CRTMQMAUTI CL command.

   The value CRLLDAP for the AUTHTYPE parameter indicates that CRLs are accessed on LDAP servers. Each authentication information object with type CRLLDAP that you create holds the address of an LDAP server. When you have more than one authentication information object, the LDAP servers to which they point *must* contain identical information. This provides continuity of service if one or more LDAP servers fail.

   z/OS Additionally, on z/OS only, all LDAP servers must be accessed using the same user ID and password. The user ID and password used are those specified in the first AUTHINFO object in the namelist.

   On all platforms, the user ID and password are sent to the LDAP server unencrypted.
2. Using the DEFINE NAMELIST MQSC command, define a namelist for the names of your authentication information objects. z/OS On z/OS, ensure that the NLTYPE namelist attribute is set to AUTHINFO.
3. Using the ALTER QMGR MQSC command, supply the namelist to the queue manager. For example:

   ```
   ALTER QMGR SSLCRLNL(sslcrlnlname)
   ```

   where `sslcrlnlname` is your namelist of authentication information objects.

This command sets a queue manager attribute called *SSLCRLNamelist*. The queue manager's initial value for this attribute is blank.

IBM i On IBM i, you can specify authentication information objects, but the queue manager uses neither authentication information objects nor a namelist of authentication information objects. Only IBM MQ clients that use a client connection table generated by an IBM i queue manager use the authentication information specified for that IBM i queue manager. The *SSLCRLNamelist* queue manager attribute on IBM i determines what authentication information such clients use. See "Accessing CRLs and ARLs on IBM i" for information about telling an IBM i queue manager how to access CRLs.

You can add up to 10 connections to alternative LDAP servers to the namelist, to ensure continuity of service if one or more LDAP servers fail. Note that the LDAP servers *must* contain identical information.

Accessing CRLs and ARLs on IBM i:

Use this procedure to access CRLs or ARLs on IBM i.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

Follow these steps to set up a CRL location for a specific certificate on IBM i:

1. Access the DCM interface, as described in "Accessing DCM" on page 661.
2. In the **Manage CRL locations** task category in the navigation panel, click **Add CRL location**. The Manage CRL Locations page is displayed in the task frame.
3. In the **CRL Location Name** field, type a CRL location name, for example `LDAP Server #1`
4. In the **LDAP Server** field, type the LDAP server name.
5. In the **Use Secure Sockets Layer (SSL)** field, select **Yes** if you want to connect to the LDAP server using SSL. Otherwise, select **No**.
6. In the **Port Number** field, type a port number for the LDAP server, for example `389`.
7. If your LDAP server does not allow anonymous users to query the directory, type a login distinguished name for the server in the **login distinguished name** field.
8. Click **OK**. DCM informs you that it has created the CRL location.
9. In the navigation panel, click **Select a Certificate Store**. The Select a Certificate Store page is displayed in the task frame.
10. Select the **Other System Certificate Store** check box and click **Continue**. The Certificate Store and Password page is displayed.
11. In the **Certificate store path and filename** field, type the IFS path and file name you set when "Creating a certificate store on IBM i" on page 663.
12. Type a password in the **Certificate Store Password** field. Click **Continue**. The Current Certificate Store page is displayed in the task frame.
13. In the **Manage Certificates** task category in the navigation panel, click **Update CRL location assignment**. The CRL Location Assignment page is displayed in the task frame.
14. Select the radio button for the CA certificate to which you want to assign the CRL location. Click **Update CRL Location Assignment**. The Update CRL Location Assignment page is displayed in the task frame.
15. Select the radio button for the CRL location which you want to assign to the certificate. Click **Update Assignment**. DCM informs you that it has updated the assignment.

Note that DCM allows you to assign a different LDAP server by Certificate Authority.

Accessing CRLs and ARLs using IBM MQ Explorer:

You can use IBM MQ Explorer to tell a queue manager how to access CRLs.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

Use the following procedure to set up an LDAP connection to a CRL:

1. Ensure that you have started your queue manager.
2. Right-click the **Authentication Information** folder and click **New -> Authentication Information**. In the property sheet that opens:
   a. On the first page **Create Authentication Information**, enter a name for the CRL(LDAP) object.
   b. On the **General** page of **Change Properties**, select the connection type. Optionally you can enter a description.
   c. Select the **CRL(LDAP)** page of **Change Properties**.
   d. Enter the LDAP server name as either the network name or the IP address.
   e. If the server requires login details, provide a user ID and if necessary a password.
   f. Click **OK**.
3. Right-click the **Namelists** folder and click **New -> Namelist**. In the property sheet that opens:
   a. Type a name for the namelist.
   b. Add the name of the CRL(LDAP) object (from step 2a) to the list.
   c. Click **OK**.
4. Right-click the queue manager, select **Properties**, and select the **SSL** page:
   a. Select the **Check certificates received by this queue manager against Certification Revocation Lists** check box.
   b. Type the name of the namelist (from step 3a) in the **CRL Namelist** field.

Accessing CRLs and ARLs with an IBM MQ MQI client

You have three options for specifying the LDAP servers that hold CRLs for checking by an IBM MQ MQI client.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

The three ways of specifying the LDAP servers are as follows:

- Using a channel definition table
- Using the SSL configuration options structure, MQSCO, on an MQCONNX call
- Using the Active Directory (on Windows systems with Active Directory support)

For more details, refer to the related information.

You can include up to 10 connections to alternative LDAP servers to ensure continuity of service if one or more LDAP servers fail. Note that the LDAP servers *must* contain identical information.

You cannot access LDAP CRLs from an IBM MQ MQI client channel running on Linux (zSeries platform).

Location of an OCSP responder, and of LDAP servers that hold CRLs:

On an IBM MQ MQI client system, you can specify the location of an OCSP responder, and of Lightweight Directory Access Protocol (LDAP) servers that hold certificate revocation lists (CRLs).

You can specify these locations in three ways, listed here in order of decreasing precedence.

IBM i (For IBM i, see Accessing CRLs and ARLs on IBM i.)

When an IBM MQ MQI client application issues an MQCONNX call

You can specify an OCSP responder or an LDAP server holding CRLs on an **MQCONNX** call.

On an **MQCONNX** call, the connect options structure, MQCNO, can reference an SSL configuration options structure, MQSCO. In turn, the MQSCO structure can reference one or more authentication information record structures, MQAIR. Each MQAIR structure contains all the information an IBM MQ MQI client requires to access an OCSP responder or an LDAP server holding CRLs. For example, one of the fields in an MQAIR structure is the URL at which a responder can be contacted. For more information about the MQAIR structure, see MQAIR - Authentication information record.

Using a client channel definition table (ccdt) to access an OCSP responder or LDAP servers

So that an IBM MQ MQI client can access an OCSP responder or LDAP servers that hold CRLs, include the attributes of one or more authentication information objects in a client channel definition table.

On a server queue manager, you can define one or more authentication information objects. The attributes of an authentication object contain all the information that is required to access an OCSP responder (on platforms where OCSP is supported) or an LDAP server that holds CRLs. One of the attributes specifies the OCSP responder URL, another specifies the host address, or IP address of a system on which an LDAP server runs.

An authentication information object with AUTHTYPE(OCSP) does not apply for use on IBM i or z/OS queue managers, but it can be specified on those platforms to be copied to the client channel definition table (CCDT) for client use.

To enable an IBM MQ MQI client to access an OCSP responder or LDAP servers that hold CRLs, the attributes of one or more authentication information objects can be included in a client channel definition table. You can include such attributes in one of the following ways:

On server platforms AIX, HP-UX, Linux, IBM i IBM i, Solaris, and Windows

You can define a namelist that contains the names of one or more authentication information objects. You can then set the queue manager attribute, **SSLCRLNameList**, to the name of this namelist.

If you are using CRLs, more than one LDAP server can be configured to provide higher availability. The intention is that each LDAP server holds the same CRLs. If one LDAP server is unavailable when it is required, an IBM MQ MQI client can attempt to access another.

The attributes of the authentication information objects identified by the namelist are referred to collectively here as the *certificate revocation location*. When you set the queue manager attribute, **SSLCRLNameList**, to the name of the namelist, the certificate revocation location is copied into the client channel definition table associated with the queue manager. If the CCDT can be accessed from a client system as a shared file, or if the CCDT is then copied to a client system, the IBM MQ MQI client on that system can use the certificate revocation location in the CCDT to access an OCSP responder or LDAP servers that hold CRLs.

If the certificate revocation location of the queue manager is changed later, the change is reflected in the CCDT associated with the queue manager. If the queue manager attribute, **SSLCRLNameList**, is set to blank, the certificate revocation location is removed from the CCDT. These changes are not reflected in any copy of the table on a client system.

If you require the certificate revocation location at the client and server ends of an MQI channel to be different, and the server queue manager is the one that is used to create the certificate revocation location, you can do it as follows:

1. On the server queue manager, create the certificate revocation location for use on the client system.
2. Copy the CCDT containing the certificate revocation location to the client system.
3. On the server queue manager, change the certificate revocation location to what is required at the server end of the MQI channel.
4. On the client machine, you can use the **runmqsc** command with the **-n** parameter.

On client platforms AIX, HP-UX, Linux, IBM i IBM i, Solaris, and Windows

You can build a CCDT on the client machine by using the runmqsc command with the **-n** parameter and **DEFINE AUTHINFO** objects in the CCDT file. The order that the objects are defined in is the order in which they are used in the file. Any name that you might use in a **DEFINE AUTHINFO** object is not retained in the file. Only positional numbers are used when you **DISPLAY** the **AUTHINFO** objects in a CCDT file.

Note: If you specify the **-n** parameter, you must not specify any other parameter.

Using Active Directory on Windows

On Windows systems, you can use the **setmqcrl** control command to publish the current CRL information in Active Directory.

Command **setmqcrl** does not publish OCSP information.

For information about this command and its syntax, see setmqcrl.

Accessing CRLs and ARLs with IBM MQ classes for Java and IBM MQ classes for JMS

IBM MQ classes for Java and IBM MQ classes for JMS access CRLs differently from other platforms.

For information about working with CRLs and ARLs with IBM MQ classes for Java, see Using certificate revocation lists

For information about working with CRLs and ARLs with IBM MQ classes for JMS, see SSLCERTSTORES object property

Manipulating authentication information objects

You can manipulate authentication information objects using MQSC or PCF commands, or the Websphere MQ Explorer.

The following MQSC commands act on authentication information objects:
- DEFINE AUTHINFO
- ALTER AUTHINFO
- DELETE AUTHINFO
- DISPLAY AUTHINFO

For a complete description of these commands, see Script (MQSC) Commands.

The following Programmable Command Format (PCF) commands act on authentication information objects:
- Create Authentication Information
- Copy Authentication Information
- Change Authentication Information
- Delete Authentication Information
- Inquire Authentication Information
- Inquire Authentication Information Names

For a complete description of these commands, see Definitions of the Programmable Command Formats.

On platforms where it is available, you can also use the IBM MQ Explorer.

Using the Pluggable Authentication Method (PAM)

UNIX Linux V 8.0.0.3

You can use PAM only on UNIX and Linux platforms. A typical UNIX system has PAM modules that implement the traditional authentication mechanism; however, there might be more. As well as the basic task of validating passwords, PAM modules can also be invoked to carry out additional rules.

Configuration files define which authentication method is to be used for each application . Example applications include the standard terminal login, ftp, and telnet.

The advantage of PAM is that the application does not need to know, or care about, how the user ID is actually being authenticated. As long as the application can provide a correct form of authentication data to PAM, the mechanism behind it is transparent.

The form of authentication data depends upon the system being used. For example, IBM MQ obtains a password through parameters, such as the MQCSP structure used in the MQCONNX API call.

Important: You cannot set the **AUTHENMD** attribute until you install IBM MQ Version 8.0.0, Fix Pack 3, and then restart the queue manager, using a **-e CMDLEVEL=***level* of *802* (on the strmqm command) to set the command level you require.

Configuring your system to use PAM

The service name used by IBM MQ, when invoking PAM, is *ibmmq*.

Note that an IBM MQ installation attempts to maintain a default PAM configuration, that permits connections from operating system users, based on known defaults for the different operating systems.

However, your system administrator must verify that rules defined in the `/etc/pam.conf`, or `/etc/pam.d/ibmmq`, files are still appropriate.

Authorizing access to objects

This section contains information about using the object authority manager and channel exit programs to control access to objects.

On UNIX, Linux, and Windows systems. you control access to objects by using the object authority manager (OAM). This collection of topics contains information about using the command interface to the OAM. It also contains a checklist you can use to determine what tasks to perform to apply security to your system, and considerations for granting users the authority to administer IBM MQ and to work with IBM MQ objects. If the supplied security mechanisms do not meet your needs, you can develop your own channel exit programs.

Controlling access to objects by using the OAM on UNIX, Linux and Windows systems

The object authority manager (OAM) provides a command interface for granting and revoking authority to IBM MQ objects.

You must be suitably authorized to use these commands, as described in “Authority to administer IBM MQ on UNIX, Linux, and Windows systems” on page 777. User IDs that are authorized to administer IBM MQ have *super user* authority to the queue manager, which means that you do not have to grant them further permission to issue any MQI requests or commands.

Giving access to an IBM MQ object on UNIX, Linux, and Windows systems

Use the **setmqaut** control command, the **SET AUTHREC** MQSC command, or the **MQCMD_SET_AUTH_REC** PCF command to give users, and groups of users, access to IBM MQ objects. Note, that on IBM MQ Appliance you can use only the **SET AUTHREC** command.

For a full definition of the **setmqaut** control command and its syntax, see setmqaut.

For a full definition of the **SET AUTHREC** MQSC command and its syntax, see SET AUTHREC.

For a full definition of the **MQCMD_SET_AUTH_REC** PCF command and its syntax, see Set Authority Record.

The queue manager must be running to use this command. When you have changed access for a principal, the changes are reflected immediately by the OAM.

To give users access to an object, you need to specify:

- The name of the queue manager that owns the objects you are working with; if you do not specify the name of a queue manager, the default queue manager is assumed.
- The name and type of the object (to identify the object uniquely). You specify the name as a *profile* ; this is either the explicit name of the object, or a generic name, including wildcard characters. For a detailed description of generic profiles, and the use of wildcard characters within them, see “Using OAM generic profiles on UNIX, Linux, and Windows systems” on page 737.
- One or more principals and group names to which the authority applies.

  If a user ID contains spaces, enclose it in quotation marks when you use this command. On Windows systems, you can qualify a user ID with a domain name. If the actual user ID contains an at sign (@) symbol, replace it with @@ to show that it is part of the user ID, not the delimiter between the user ID and the domain name.

- A list of authorizations. Each item in the list specifies a type of access that is to be granted to that object (or revoked from it). Each authorization in the list is specified as a keyword, prefixed with a plus sign (+) or a minus sign (-). Use a plus sign to add the specified authorization, and a minus sign to remove the authorization. There must be no spaces between the + or - sign and the keyword.

  You can specify any number of authorizations in a single command. For example, the list of authorizations to permit a user or group to put messages on a queue and to browse them, but to revoke access to get messages is:

  ```
  +browse -get +put
  ```

Examples of using the `setmqaut` command

The following examples show how to use the **setmqaut** command to grant and revoke permission to use an object:

```
setmqaut -m saturn.queue.manager -t queue -n RED.LOCAL.QUEUE
        -g groupa +browse -get +put
```

In this example:
- `saturn.queue.manager` is the queue manager name
- `queue` is the object type
- `RED.LOCAL.QUEUE` is the object name
- `groupa` is the identifier of the group with authorizations that are to change
- `+browse -get +put` is the authorization list for the specified queue
  - `+browse` adds authorization to browse messages on the queue (to issue **`MQGET`** with the browse option)
  - `-get` removes authorization to get (**`MQGET`**) messages from the queue
  - `+put` adds authorization to put (**`MQPUT`**) messages on the queue

The following command revokes put authority on the queue MyQueue from principal fvuser and from groups groupa and groupb. On UNIX and Linux systems, this command also revokes put authority for all principals in the same primary group as fvuser.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -p fvuser
        -g groupa -g groupb -put
```

Using the `setmqaut` command with a different authorization service

If you are using your own authorization service instead of the OAM, you can specify the name of this service on the **`setmqaut`** command to direct the command to this service. You must specify this parameter if you have multiple installable components running at the same time; if you do not, the update is made to the first installable component for the authorization service. By default, this is the supplied OAM.

Usage notes for SET AUTHREC

The list of authorizations to add and the list of authorizations to remove must not overlap. For example, you cannot add display authority and remove display authority with the same command. This rule applies even if the authorities are expressed using different options. For example, the following command fails because DSP authority overlaps with ALLADM authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALLADM)
```

The exception to this overlap behavior is with the ALL authority. The following command first adds ALL authorities then removes the SETID authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(ALL) AUTHRMV(SETID)
```

The following command first removes ALL authorities then adds the DSP authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALL)
```

Regardless of the order in which they are provided on the command, the ALL are processed first.

Using OAM generic profiles on UNIX, Linux, and Windows systems

Use OAM generic profiles to set, in a single operation, a user's privileges for many objects; rather than having to issue separate **setmqaut** commands, or **SET AUTHREC** commands, against each individual object when it is created. Note, that on IBM MQ Appliance you can use only the **SET AUTHREC** command.

Using generic profiles in the setmqaut or SET AUTHREC commands, enables you to set a generic authority for all objects that fit that profile.

This collection of topics describes the use of generic profiles in more detail.

Using wildcard characters in OAM profiles

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the question mark (?) wildcard character matches any single character in a name. So, if you specify `ABC.?EF`, the authorization you give to that profile applies to any objects with the names `ABC.DEF`, `ABC.CEF`, `ABC.BEF`, and so on.

The wildcard characters available are:

? Use the question mark (?) instead of any single character. For example, `AB.?D` applies to the objects `AB.CD`, `AB.ED`, and `AB.FD`.

* Use the asterisk (*) as:

- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in `ABC.DEF.GHI`, the qualifiers are `ABC`, `DEF`, and `GHI`.

  For example, `ABC.*.JKL` applies to the objects `ABC.DEF.JKL`, and `ABC.GHI.JKL`. (Note that it does **not** apply to `ABC.JKL` ; * used in this context always indicates one qualifier.)
- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, `ABC.DE*.JKL` applies to the objects `ABC.DE.JKL`, `ABC.DEF.JKL`, and `ABC.DEGH.JKL`.

** Use the double asterisk (**) **once** in a profile name as:

- The entire profile name to match all object names. For example if you use `-t prcs` to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, `**.ABC` identifies all objects with the final qualifier ABC.

Note: When using wildcard characters on UNIX and Linux systems, you **must** enclose the profile name in single quotation marks.

Profile priorities

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
setmqaut -n AB.* -t q +put -p fred
setmqaut -n AB.C* -t q +get -p fred
```

The first gives put authority to all queues for the principal fred with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either setmqaut could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific then a generic character. So, in this example, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:
1. ?
2. *
3. **

Dumping profile settings

For a full definition of the **dmpmqaut** control command and its syntax, see dmpmqaut.

For a full definition of the **DISPLAY AUTHREC** MQSC command and its syntax, see DISPLAY AUTHREC.

For a full definition of the **MQCMD_INQUIRE_AUTH_RECS** PCF command and its syntax, see Inquire Authority Records.

The following examples show the use of the **dmpmqaut** control command to dump authority records for generic profiles:
1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.
   ```
   dmpmqaut -m qm1 -n a.b.c -t q -p user1
   ```
   The resulting dump looks something like this:
   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```
   Note: Although UNIX and Linux users can use the `-p` option for the **dmpmqaut** command, they must use `-g groupname` instead when defining authorizations.
2. This example dumps all authority records with a profile that matches queue a.b.c.
   ```
   dmpmqaut -m qmgr1 -n a.b.c -t q
   ```
   The resulting dump looks something like this:
   ```
   profile:     a.b.c
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   - - - - - - - - - - - - - - - - -
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   - - - - - - - - - - - - - - - - -
   profile:     a.**
   object type: queue
   entity:      group1
   type:        group
   authority:   get
   ```
3. This example dumps all authority records for profile a.b.*, of type queue.

```
dmpmqaut -m qmgr1 -n a.b.* -t q
```

The resulting dump looks something like this:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

4. This example dumps all authority records for queue manager qmX.

```
dmpmqaut -m qmX
```

The resulting dump looks something like this:

```
profile:     q1
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - - - -
profile:     q*
object type: queue
entity:      user1
type:        principal
authority:   get, browse
- - - - - - - - - - - - - - - - - -
profile:     name.*
object type: namelist
entity:      user2
type:        principal
authority:   get
- - - - - - - - - - - - - - - - - -
profile:     pr1
object type: process
entity:      group1
type:        group
authority:   get
```

5. This example dumps all profile names and object types for queue manager qmX.

```
dmpmqaut -m qmX -l
```

The resulting dump looks something like this:

```
profile: q1, type: queue
profile: q*, type: queue
profile: name.*, type: namelist
profile: pr1, type: process
```

Note: For IBM MQ for Windows only, all principals displayed include domain information, for example:

```
profile:     a.b.*
object type: queue
entity:      user1@domain1
type:        principal
authority:   get, browse, put, inq
```

Using wildcard characters in OAM profiles

Use wildcard characters in an object authority manager (OAM) profile name to make that profile applicable to more than one object.

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the question mark (?) wildcard character matches any single character in a name. So, if you specify `ABC.?EF`, the authorization you give to that profile applies to any objects with the names `ABC.DEF`, `ABC.CEF`, `ABC.BEF`, and so on.

The wildcard characters available are:

?
: Use the question mark (?) instead of any single character. For example, `AB.?D` applies to the objects `AB.CD`, `AB.ED`, and `AB.FD`.

*
: Use the asterisk (*) as:
- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in `ABC.DEF.GHI`, the qualifiers are `ABC`, `DEF`, and `GHI`.

  For example, `ABC.*.JKL` applies to the objects `ABC.DEF.JKL`, and `ABC.GHI.JKL`. (Note that it does **not** apply to `ABC.JKL` ; * used in this context always indicates one qualifier.)
- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, `ABC.DE*.JKL` applies to the objects `ABC.DE.JKL`, `ABC.DEF.JKL`, and `ABC.DEGH.JKL`.

**
: Use the double asterisk (**) **once** in a profile name as:
- The entire profile name to match all object names. For example if you use `-t prcs` to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, `**.ABC` identifies all objects with the final qualifier ABC.

Note: When using wildcard characters on UNIX and Linux systems, you **must** enclose the profile name in single quotation marks.

Profile priorities

More than one generic profile can apply to a single object. Where this is the case, the most specific rule applies.

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
setmqaut -n AB.* -t q +put -p fred
setmqaut -n AB.C* -t q +get -p fred
```

The first gives put authority to all queues for the principal fred with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either setmqaut could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific then a generic character. So, in this example, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:

1. ?
2. *
3. **

See SET AUTHREC for the equivalent information when using this MQSC command.

Dumping profile settings

Use the **dmpmqaut** control command, the **DISPLAY AUTHREC** MQSC command, or the **MQCMD_INQUIRE_AUTH_RECS** PCF command to dump the current authorizations associated with a specified profile. Note, that on IBM MQ Appliance you can use only the **DISPLAY AUTHREC** command.

For a full definition of the **dmpmqaut** control command and its syntax, see dmpmqaut.

For a full definition of the **DISPLAY AUTHREC** MQSC command and its syntax, see DISPLAY AUTHREC.

For a full definition of the **MQCMD_INQUIRE_AUTH_RECS** PCF command and its syntax, see Inquire Authority Records.

The following examples show the use of the **dmpmqaut** control command to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.

   ```
   dmpmqaut -m qm1 -n a.b.c -t q -p user1
   ```

   The resulting dump looks something like this example:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```

   Note: UNIX and Linux users cannot use the `-p` option; they must use `-g groupname` instead.
2. This example dumps all authority records with a profile that matches queue a.b.c.

   ```
   dmpmqaut -m qmgr1 -n a.b.c -t q
   ```

   The resulting dump looks something like this example:

   ```
   profile:     a.b.c
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   - - - - - - - - - - - - - - - - -
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   - - - - - - - - - - - - - - - - -
   profile:     a.**
   object type: queue
   entity:      group1
   type:        group
   authority:   get
   ```
3. This example dumps all authority records for profile a.b.*, of type queue.

   ```
   dmpmqaut -m qmgr1 -n a.b.* -t q
   ```

   The resulting dump looks something like this example:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

4. This example dumps all authority records for queue manager qmX.

   ```
   dmpmqaut -m qmX
   ```

   The resulting dump looks something like this example:

   ```
   profile:     q1
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   - - - - - - - - - - - - - - - - -
   profile:     q*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse
   - - - - - - - - - - - - - - - - -
   profile:     name.*
   object type: namelist
   entity:      user2
   type:        principal
   authority:   get
   - - - - - - - - - - - - - - - - -
   profile:     pr1
   object type: process
   entity:      group1
   type:        group
   authority:   get
   ```

5. This example dumps all profile names and object types for queue manager qmX.

   ```
   dmpmqaut -m qmX -l
   ```

   The resulting dump looks something like this example:

   ```
   profile: q1, type: queue
   profile: q*, type: queue
   profile: name.*, type: namelist
   profile: pr1, type: process
   ```

Note: For IBM MQ for Windows only, all principals displayed include domain information, for example:

```
profile:     a.b.*
object type: queue
entity:      user1@domain1
type:        principal
authority:   get, browse, put, inq
```

Displaying access settings

Use the **dspmqaut** control command, the **DISPLAY AUTHREC** MQSC command, or the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command to view the authorizations that a specific principal or group has for a particular object. Note, that on IBM MQ Appliance you can use only the **DISPLAY AUTHREC** command.

The queue manager must be running to use this command. When you change access for a principal, the changes are reflected immediately by the OAM. Authorization can be displayed for only one group or principal at a time.

For a full definition of the **dmpmqaut** control command and its syntax, see dmpmqaut.

For a full definition of the **DISPLAY AUTHREC** MQSC command and its syntax, see DISPLAY AUTHREC.

For a full definition of the **MQCMD_INQUIRE_AUTH_RECS** PCF command and its syntax, see Inquire Authority Records.

The following example shows the use of the **dspmqaut** control command to display the authorizations that the group `GpAdmin` has to a process definition named `Annuities` that is on queue manager `QueueMan1`.

```
dspmqaut -m QueueMan1 -t process -n Annuities -g GpAdmin
```

Changing and revoking access to an IBM MQ object

To change the level of access that a user or group has to an object, use the **setmqaut** control command, the **DELETE AUTHREC** MQSC command, or the **MQCMD_DELETE_AUTH_REC** PCF command. Note, that on IBM MQ Appliance you can use only the **DELETE AUTHREC** command.

The process of removing the user from a group is described in:
- “Creating and managing groups on Windows” on page 512
- “Creating and managing groups on HP-UX” on page 514
- “Creating and managing groups on AIX” on page 515
- “Creating and managing groups on Solaris” on page 517
- “Creating and managing groups on Linux” on page 517

.

The user ID that creates an IBM MQ object is granted full control authorities to that object. If you remove this user ID from the local mqm group (or the Administrators group on Windows systems) these authorities are not revoked. Use the **setmqaut** control command or the **MQCMD_DELETE_AUTH_REC** PCF command to revoke access to an object for the user ID that created it, after removing it from the mqm or Administrators group.

For a full definition of the setmqaut control command and its syntax, see setmqaut.

For a full definition of the **DELETE AUTHREC** MQSC command and its syntax, see DELETE AUTHREC.

For a full definition of the **MQCMD_DELETE_AUTH_REC** PCF command and its syntax, see Delete Authority Record.

On Windows, from IBM MQ Version 8.0, you can delete the OAM entries corresponding to a particular Windows user account at any time using the **-u** *SID* parameter of **setmqaut**.

Prior to IBM MQ Version 8.0, you had to delete the OAM entries corresponding to a particular Windows user account before deleting the user profile. It was impossible to remove the OAM entries after removing the user account.

Preventing security access checks on Windows, UNIX and Linux systems

To turn off all security checking you can disable the OAM. This might be suitable for a test environment. Having disabled or removed the OAM, you cannot add an OAM to an existing queue manager.

If you decide that you do not want to perform security checks (for example, in a test environment), you can disable the OAM in one of two ways:

- Before you create a queue manager, set the operating system environment variable MQSNOAUT (if you do this, you cannot add an OAM later):

  See Environment variables for information about the implications of setting the MQSNOAUT variable, and how you set MQSNOAUT on Windows and UNIX platforms.
- Edit the queue manager configuration file to remove the service. (If you do this, you cannot add an OAM later.)

If you use setmqaut, or dspmqaut while the OAM is disabled, note the following points:

- The OAM does not validate the specified principal, or group, meaning that the command can accept invalid values.
- The OAM does not perform security checks and indicates that all principals and groups are authorized to perform all applicable object operations.

When an OAM is removed, it cannot be put back on an existing queue manager. This is because the OAM needs to be in place at object creation time. To use the IBM MQ OAM again after it has been removed, the queue manager needs to be rebuilt.

Related information:

Installable services

Granting required access to resources

Use this topic to determine what tasks to perform to apply security to your IBM MQ system.

About this task

During this task, you decide what actions are necessary to apply the appropriate level of security to the elements of your IBM MQ installation. Each individual task you are referred to gives step-by-step instructions for all platforms.

Procedure

1. Do you need to limit access to your queue manager to certain users?
   a. No: Take no further action.
   b. Yes: Go to the next question.
2. Do these users need partial administrative access on a subset of queue manager resources?
   a. No: Go to the next question.
   b. Yes: See “Granting partial administrative access on a subset of queue manager resources” on page 745.
3. Do these users need full administrative access on a subset of queue manager resources?
   a. No: Go to the next question.
   b. Yes: See “Granting full administrative access on a subset of queue manager resources” on page 753.
4. Do these users need read only access to all queue manager resources?
   a. No: Go to the next question.

   b. Yes: See “Granting read-only access to all resources on a queue manager” on page 757.
5. Do these users need full administrative access on all queue manager resources?
   a. No: Go to the next question.
   b. Yes: See “Granting full administrative access to all resources on a queue manager” on page 759.
6. Do you need user applications to connect to your queue manager?
   a. No: Disable connectivity, as described in “Removing connectivity to the queue manager” on page 760
   b. Yes: See “Allowing user applications to connect to your queue manager” on page 760.

Granting partial administrative access on a subset of queue manager resources

You need to give certain users partial administrative access to some, but not all, queue manager resources. Use this table to determine the actions you need to take.

Table 79. Granting partial administrative access to a subset of queue manager resources

The users need to administer objects of this type	Perform this action
Queues	Grant partial administrative access to the required queues, as described in “Granting limited administrative access to some queues” on page 746
Topics	Grant partial administrative access to the required topics, as described in “Granting limited administrative access to some topics” on page 747
Channels	Grant partial administrative access to the required channels, as described in “Granting limited administrative access to some channels” on page 748
The queue manager	Grant partial administrative access to the queue manager, as described in “Granting limited administrative access to a queue manager” on page 749
Processes	Grant partial administrative access to the required processes, as described in “Granting limited administrative access to some processes” on page 750
Namelists	Grant partial administrative access to the required namelists, as described in “Granting limited administrative access to some namelists” on page 751
Services	Grant partial administrative access to the required services, as described in “Granting limited administrative access to some services” on page 752

Granting limited administrative access to some queues

Grant partial administrative access to some queues on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some queues for some actions, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName ReqdAction
  ```

- IBM i For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(ReqdAction) MQMNAME(' QMgrName ')
  ```

- z/OS For z/OS, issue the following commands: z/OS

  ```
  RDEFINE MQADMIN QMgrName.QUEUE. ObjectProfile UACC(NONE)
  PERMIT QMgrName.QUEUE. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  These commands grant access to the specified queue. To determine which MQSC commands the user can perform on the queue, issue the following commands for each MQSC command:

  ```
  RDEFINE MQCMDS QMgrName. ReqdAction. QType UACC(NONE)
  PERMIT QMgrName. ReqdAction. QType CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
  ```

  To permit the user to use the DISPLAY QUEUE command, issue the following commands:

  ```
  RDEFINE MQCMDS QMgrName.DISPLAY. QType UACC(NONE)
  PERMIT QMgrName.DISPLAY. QType CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

  ReqdAction
  The action you are allowing the group to take:
  - On UNIX, Linux and Windows systems, any combination of the following authorizations: +chg, +clr, +dlt, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.
  - IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMDLT, *ADMDSP. The authorization *ALLADM is equivalent to all these individual authorizations.
  - z/OS On z/OS, one of the values ALTER, CLEAR, DELETE, or MOVE.

  Note: Granting +crt for queues indirectly makes the user or group an administrator. Do not use +crt authority to grant limited administrative access to some queues.

QType

For the DISPLAY command, one of the values QUEUE, QLOCAL, QALIAS, QMODEL, QREMOTE, or QCLUSTER.

For other values of *ReqdAction*, one of the values QLOCAL, QALIAS, QMODEL, or QREMOTE.

Granting limited administrative access to some topics

Grant partial administrative access to some topics on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some topics for some actions, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName ReqdAction
  ```

- IBM i For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(ReqdAction) MQMNAME(' QMgrName ')
  ```

- z/OS For z/OS, issue the following commands: z/OS

  ```
  RDEFINE MQADMIN QMgrName.TOPIC. ObjectProfile UACC(NONE)
  PERMIT QMgrName.TOPIC. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  These commands grant access to the specified topic. To determine which MQSC commands the user can perform on the topic, issue the following commands for each MQSC command:

  ```
  RDEFINE MQCMDS QMgrName. ReqdAction.TOPIC UACC(NONE)
  PERMIT QMgrName. ReqdAction.TOPIC CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
  ```

  To permit the user to use the DISPLAY TOPIC command, issue the following commands:

  ```
  RDEFINE MQCMDS QMgrName.DISPLAY.TOPIC UACC(NONE)
  PERMIT QMgrName.DISPLAY.TOPIC CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName

  The name of the queue manager. z/OS On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile

  The name of the object or generic profile for which to change authorizations.

  GroupName

  The name of the group to be granted access.

  ReqdAction

  The action you are allowing the group to take:

  – On UNIX, Linux and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. +ctrl. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL. The authorization *ALLADM is equivalent to all these individual authorizations.
- z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to some channels

Grant partial administrative access to some channels on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some channels for some actions, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:

```
setmqaut -m QMgrName -n ObjectProfile -t channel -g GroupName ReqdAction
```

- IBM i For IBM i, issue the following command:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*CHL) USER(GroupName) AUT(ReqdAction) MQMNAME(' QMgrName ')
```

- z/OS For z/OS, issue the following commands: z/OS

```
RDEFINE MQADMIN QMgrName.CHANNEL. ObjectProfile UACC(NONE)
PERMIT QMgrName.CHANNEL. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

  These commands grant access to the specified channel. To determine which MQSC commands the user can perform on the channel, issue the following commands for each MQSC command:

```
RDEFINE MQCMDS QMgrName. ReqdAction.CHANNEL UACC(NONE)
PERMIT QMgrName. ReqdAction.CHANNEL CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

  To permit the user to use the DISPLAY CHANNEL command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.DISPLAY.CHANNEL UACC(NONE)
PERMIT QMgrName.DISPLAY.CHANNEL CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

  ReqdAction
  The action you are allowing the group to take:

  - On UNIX, Linux and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. +ctrl, +ctrlx. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

- – IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL, *CTRLx. The authorization *ALLADM is equivalent to all these individual authorizations.
- – z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to a queue manager

Grant partial administrative access to a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to perform some actions on the queue manager, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t qmgr -g GroupName ReqdAction
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*MQM) USER(GroupName) AUT(ReqdAction) MQMNAME(' QMgrName ')
  ```

Results

To determine which MQSC commands the user can perform on the queue manager, issue the following commands for each MQSC command:

```
RDEFINE MQCMDS QMgrName. ReqdAction.QMGR UACC(NONE)
PERMIT QMgrName. ReqdAction.QMGR CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

To permit the user to use the DISPLAY QMGR command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.DISPLAY.QMGR UACC(NONE)
PERMIT QMgrName.DISPLAY.QMGR CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

ReqdAction
The action you are allowing the group to take:

- On UNIX, Linux and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

  Although +set is an MQI authorization and not normally considered administrative, granting +set on the queue manager can indirectly lead to full administrative authority. Do not grant +set to ordinary users and applications.

- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP. The authorization *ALLADM is equivalent to all these individual authorizations.

Granting limited administrative access to some processes

Grant partial administrative access to some processes on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some processes for some actions, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t process -g GroupName ReqdAction
  ```
- IBM i For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*PRC) USER(GroupName) AUT(ReqdAction) MQMNAME(' QMgrName ')
  ```
- z/OS For z/OS, issue the following commands: z/OS
  ```
  RDEFINE MQADMIN QMgrName.PROCESS. ObjectProfile UACC(NONE)
  PERMIT QMgrName.PROCESS. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  These commands grant access to the specified channel. To determine which MQSC commands the user can perform on the channel, issue the following commands for each MQSC command:
  ```
  RDEFINE MQCMDS QMgrName. ReqdAction.PROCESS UACC(NONE)
  PERMIT QMgrName. ReqdAction.PROCESS CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
  ```

  To permit the user to use the DISPLAY PROCESS command, issue the following commands:
  ```
  RDEFINE MQCMDS QMgrName.DISPLAY.PROCESS UACC(NONE)
  PERMIT QMgrName.DISPLAY.PROCESS CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. z/OS On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

  ReqdAction
  The action you are allowing the group to take:
  - On UNIX, Linux and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.
  - IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP. The authorization *ALLADM is equivalent to all these individual authorizations.

  - z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to some namelists

Grant partial administrative access to some namelists on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some namelists for some actions, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:

```
setmqaut -m QMgrName -n ObjectProfile -t namelist -g GroupName ReqdAction
```

- IBM i For IBM i, issue the following command:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*NMLIST) USER(GroupName) AUT(ReqdAction) MQMNAME(' QMgrName ')
```

- z/OS For z/OS, issue the following commands: z/OS

```
RDEFINE MQADMIN QMgrName.NAMELIST. ObjectProfile UACC(NONE)
PERMIT QMgrName.NAMELIST. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

  z/OS These commands grant access to the specified namelist. To determine which MQSC commands the user can perform on the namelist, issue the following commands for each MQSC command: z/OS

```
RDEFINE MQCMDS QMgrName. ReqdAction.NAMELIST UACC(NONE)
PERMIT QMgrName. ReqdAction.NAMELIST CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

  z/OS To permit the user to use the DISPLAY NAMELIST command, issue the following commands: z/OS

```
RDEFINE MQCMDS QMgrName.DISPLAY.NAMELIST UACC(NONE)
PERMIT QMgrName.DISPLAY.NAMELIST CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. z/OS On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

  ReqdAction
  The action you are allowing the group to take:
  - On UNIX, Linux and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +ctrl, +ctrlx, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL, *CTRLX. The authorization *ALLADM is equivalent to all these individual authorizations.
- z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to some services

Grant partial administrative access to some services on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some services for some actions, use the appropriate commands for your operating system.

Note: Service objects do not exist on z/OS.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:

```
setmqaut -m QMgrName -n ObjectProfile -t service -g GroupName ReqdAction
```

- For IBM i, issue the following command:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*SVC) USER(GroupName) AUT(ReqdAction) MQMNAME(' QMgrName ')
```

Results

These commands grant access to the specified service. To determine which MQSC commands the user can perform on the service, issue the following commands for each MQSC command:

```
RDEFINE MQCMDS QMgrName. ReqdAction.SERVICE UACC(NONE)
PERMIT QMgrName. ReqdAction.SERVICE CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

To permit the user to use the DISPLAY SERVICE command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.DISPLAY.SERVICE UACC(NONE)
PERMIT QMgrName.DISPLAY.SERVICE CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

ReqdAction
The action you are allowing the group to take:

- On UNIX, Linux and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +ctrl, +ctrlx, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL, *CTRLX. The authorization *ALLADM is equivalent to all these individual authorizations.

Granting full administrative access on a subset of queue manager resources

You need to give certain users full administrative access to some, but not all, queue manager resources. Use these tables to determine the actions you need to take.

Table 80. Granting full administrative access to a subset of queue manager resources

The users need to administer objects of this type	Perform this action
Queues	Grant full administrative access to the required queues, as described in “Granting full administrative access to some queues”
Topics	Grant full administrative access to the required topics, as described in “Granting full administrative access to some topics” on page 754
Channels	Grant full administrative access to the required channels, as described in “Granting full administrative access to some channels” on page 754
The queue manager	Grant full administrative access to the queue manager, as described in “Granting full administrative access to a queue manager” on page 755
Processes	Grant full administrative access to the required processes, as described in “Granting full administrative access to some processes” on page 756
Namelists	Grant full administrative access to the required namelists, as described in “Granting full administrative access to some namelists” on page 756
Services	Grant full administrative access to the required services, as described in “Granting full administrative access to some services” on page 757

Granting full administrative access to some queues

Grant full administrative access to some queues on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some queues, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
```
setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +alladm
```
- For IBM i, issue the following command:
```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*ALLADM) MQMNAME(' QMgrName ')
```
- For z/OS, issue the following commands:

```
RDEFINE MQADMIN QMgrName.QUEUE. ObjectProfile UACC(NONE)
PERMIT QMgrName.QUEUE. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting full administrative access to some topics

Grant full administrative access to some topics on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some topics for some actions, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName +alladm
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(ALLADM) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQADMIN QMgrName.TOPIC. ObjectProfile UACC(NONE)
  PERMIT QMgrName.TOPIC. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting full administrative access to some channels

Grant full administrative access to some channels on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some channels, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t channel -g GroupName +alladm
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*CHL) USER(GroupName) AUT(ALLADM) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQADMIN QMgrName.CHANNEL. ObjectProfile UACC(NONE)
  PERMIT QMgrName.CHANNEL. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting full administrative access to a queue manager

Grant full administrative access to a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to the queue manager, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -t qmgr -g GroupName +alladm
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*MQM) USER(GroupName) AUT(*ALLADM) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQADMIN QMgrName.QMGR UACC(NONE)
  PERMIT QMgrName.QMGR CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting full administrative access to some processes

Grant full administrative access to some processes on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some processes, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t process -g GroupName +alladm
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*PRC) USER(GroupName) AUT(*ALLADM) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQADMIN QMgrName.CHANNEL. ObjectProfile UACC(NONE)
  PERMIT QMgrName.PROCESS. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting full administrative access to some namelists

Grant full administrative access to some namelists on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some namelists, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t namelist -g GroupName +alladm
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*NMLIST) USER(GroupName) AUT(*ALLADM) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQADMIN QMgrName.NAMELIST. ObjectProfile UACC(NONE)
  PERMIT QMgrName.NAMELIST. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting full administrative access to some services

Grant full administrative access to some services on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some services, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
```
setmqaut -m QMgrName -n ObjectProfile -t service -g GroupName +alladm
```
- For IBM i, issue the following command:
```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*SVC) USER(GroupName) AUT(*ALLADM) MQMNAME(' QMgrName ')
```
- For z/OS, issue the following commands:
```
RDEFINE MQADMIN QMgrName.SERVICE. ObjectProfile UACC(NONE)
PERMIT QMgrName.SERVICE. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting read-only access to all resources on a queue manager

Grant read-only access to all the resources on a queue manager, to each user or group of users with a business need for it.

About this task

Use the Add Role Based Authorities wizard or the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- Using the wizard:
  1. In the IBM MQ Explorer Navigator pane, right-click the queue manager and click **Object Authorities** > **Add Role Based Authorities** The Add Role Based Authorities wizard opens.
- For UNIX and Windows systems, issue the following commands:

```
setmqaut -m QMgrName -n ** -t queue -g GroupName +browse +dsp
setmqaut -m QMgrName -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -g GroupName +dsp +inq +put
setmqaut -m QMgrName -n SYSTEM.MQEXPLORER.REPLY.MODEL -t queue -g GroupName +dsp +inq +get
setmqaut -m QMgrName -n ** -t topic -g GroupName +dsp
setmqaut -m QMgrName -n ** -t channel -g GroupName +dsp +inq
setmqaut -m QMgrName -n ** -t clntconn -g GroupName +dsp
setmqaut -m QMgrName -n ** -t authinfo -g GroupName +dsp
setmqaut -m QMgrName -n ** -t listener -g GroupName +dsp
setmqaut -m QMgrName -n ** -t namelist -g GroupName +dsp
setmqaut -m QMgrName -n ** -t process -g GroupName +dsp
setmqaut -m QMgrName -n ** -t service -g GroupName +dsp
setmqaut -m QMgrName -t qmgr -g GroupName +dsp +inq +connect
```

  The specific authorities to SYSTEM.ADMIN.COMMAND.QUEUE and SYSTEM.MQEXPLORER.REPLY.MODEL are necessary only if you want to use the MQ Explorer.
- For IBM i, issue the following commands:

```
GRTMQMAUT OBJ(*ALL) OBJTYPE(*Q) USER('GroupName') AUT(*ADMDSP *BROWSE) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*TOPIC) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*CHL) USER('GroupName') AUT(*ADMDSP *INQ) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*CLTCN) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*AUTHINFO) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*LSR) USER('GroupName') AUT(*ADMDSP)MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*NMLIST) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*PRC) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*SVC) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ('object-name') OBJTYPE(*MQM) USER('GroupName') AUT(*ADMDSP *CONNECT *INQ) MQMNAME('QMgrName')
```

- For z/OS, issue the following commands:

```
RDEFINE MQQUEUE QMgrName.** UACC(NONE)
PERMIT QMgrName.** CLASS(MQQUEUE) ID(GroupName) ACCESS(READ)
RDEFINE MQTOPIC QMgrName.** UACC(NONE)
PERMIT QMgrName.** CLASS(MQTOPIC) ID(GroupName) ACCESS(READ)
RDEFINE MQPROC QMgrName.** UACC(NONE)
PERMIT QMgrName.** CLASS(MQPROC) ID(GroupName) ACCESS(READ)
RDEFINE MQNLIST QMgrName.** UACC(NONE)
PERMIT QMgrName.** CLASS(MQNLIST) ID(GroupName) ACCESS(READ)
RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
PERMIT QMgrName.BATCH CLASS(MQCONN) ID(GroupName) ACCESS(READ)
RDEFINE MQCONN QMgrName.CICS UACC(NONE)
PERMIT QMgrName.CICS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
RDEFINE MQCONN QMgrName.IMS UACC(NONE)
PERMIT QMgrName.IMS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
RDEFINE MQCONN QMgrName.CHIN UACC(NONE)
PERMIT QMgrName.CHIN CLASS(MQCONN) ID(GroupName) ACCESS(READ)
```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  GroupName
  The name of the group to be granted access.

Granting full administrative access to all resources on a queue manager

Grant full administrative access to all the resources on a queue manager, to each user or group of users with a business need for it.

About this task

Use the Add Role Based Authorities wizard or the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- Using the wizard:
  1. In the IBM MQ Explorer Navigator pane, right-click the queue manager and click **Object Authorities** > **Add Role Based Authorities** The Add Role Based Authorities wizard opens.
- For UNIX and Linux systems, issue the following commands:

```
setmqaut -m QMgrName -n '**' -t queue -g GroupName +alladm +browse
setmqaut -m QMgrName -n @class -t queue -g GroupName +crt
setmqaut -m QMgrName -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -g GroupName +dsp +inq +put
setmqaut -m QMgrName -n SYSTEM.MQEXPLORER.REPLY.QUEUE -t queue -g GroupName +dsp +inq +get
setmqaut -m QMgrName -n '**' -t topic -g GroupName +alladm
setmqaut -m QMgrName -n @class -t topic -g GroupName +crt
setmqaut -m QMgrName -n '**' -t channel -g GroupName +alladm
setmqaut -m QMgrName -n @class -t channel -g GroupName +crt
setmqaut -m QMgrName -n '**' -t clntconn -g GroupName +alladm
setmqaut -m QMgrName -n @class -t clntconn -g GroupName +crt
setmqaut -m QMgrName -n '**' -t authinfo -g GroupName +alladm
setmqaut -m QMgrName -n @class -t authinfo -g GroupName +crt
setmqaut -m QMgrName -n '**' -t listener -g GroupName +alladm
setmqaut -m QMgrName -n @class -t listener -g GroupName +crt
setmqaut -m QMgrName -n '**' -t namelist -g GroupName +alladm
setmqaut -m QMgrName -n @class -t namelist -g GroupName +crt
setmqaut -m QMgrName -n '**' -t process -g GroupName +alladm
setmqaut -m QMgrName -n @class -t process -g GroupName +crt
setmqaut -m QMgrName -n '**' -t service -g GroupName +alladm
setmqaut -m QMgrName -n @class -t service -g GroupName +crt
setmqaut -m QMgrName -t qmgr -g GroupName +alladm +conn
```

- For Windows systems, issue the same commands as for UNIX and Linux systems, but using the profile name @CLASS instead of @class.
- For IBM i, issue the following command:

```
GRTMQMAUT OBJ(*ALL) OBJTYPE(*ALL) USER(' GroupName ') AUT(*ALLADM) MQMNAME(' QMgrName ')
```

- For z/OS, issue the following commands:

```
RDEFINE MQADMIN QMgrName.*.** UACC(NONE)
PERMIT QMgrName.*.** CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  GroupName
  The name of the group to be granted access.

Removing connectivity to the queue manager

If you do not want user applications to connect to your queue manager, remove their authority to connect to it.

About this task

Revoke the authority of all users to connect to the queue manager by using the appropriate command for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the DELETE AUTHREC command.

Note: On IBM MQ Appliance you can use only the **DELETE AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -t qmgr -g GroupName -connect
  ```
- For IBM i, issue the following command:
  ```
  RVKMQMAUT OBJ (' QMgrName ') OBJTYPE(*MQM) USER(*ALL) AUT(*CONNECT)
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
  RDEFINE MQCONN QMgrName.CHIN UACC(NONE)
  RDEFINE MQCONN QMgrName.CICS UACC(NONE)
  RDEFINE MQCONN QMgrName.IMS UACC(NONE)
  ```
  Do not issue any PERMIT commands. The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  GroupName
  The name of the group to be denied access.

Allowing user applications to connect to your queue manager

You want to allow user application to connect to your queue manager. Use the tables in this topic to determine what actions to take.

First, determine whether client applications will connect to your queue manager.

If none of the applications that will connect to your queue manager are client applications, disable remote access as described in “Disabling remote access to the queue manager” on page 768.

If one or more of the applications that will connect to your queue manager are client applications, secure remote connectivity as described in “Securing remote connectivity to the queue manager” on page 761.

In both cases, set up connection security as described in “Setting up connection security” on page 768

If you want to control access to resources for each user connecting to the queue manager, see the following table. If the statement in the first column is true, take the action listed in the second column.

Statement	Take this action
You have applications that make use of queues	See "Controlling user access to queues" on page 769
You have applications that make use of topics	See "Controlling user access to topics" on page 773.
You have applications that inquire on the queue manager object	See "Granting authority to inquire on a queue manager" on page 775.
You have applications that use process objects	See "Granting authority to access processes" on page 776
You have applications that make use of namelists	See "Granting authority to access namelists" on page 776

Securing remote connectivity to the queue manager

You can secure remote connectivity to the queue manager using SSL or TLS, a security exit, channel authentication records, or a combination of these methods.

About this task

You connect a client to the queue manager by using a client-connection channel on the client workstation and a server-connection channel on the server. Secure such connections in one of the following ways.

Procedure

1. Using SSL or TLS with channel authentication records:
   a. Prevent any Distinguished Name (DN) from opening a channel, by using an SSLPEERMAP channel authentication record to map all DNs to USERSRC(NOACCESS).
   b. Allow specific DNs or sets of DNs to open a channel by using an SSLPEERMAP channel authentication record to map them to USERSRC(CHANNEL).
2. Using SSL or TLS with a security exit:
   a. Set MCAUSER on the server-connection channel to a user identifier with no privileges.
   b. Write a security exit to assign an MCAUSER value depending on the value of SSL DN it receives in the SSLPeerNamePtr and SSLPeerNameLength fields passed to the exit in the MQCD structure.
3. Using SSL or TLS with fixed channel definition values:
   a. Set SSLPEER on the server-connection channel to a specific value or narrow range of values.
   b. Set MCAUSER on the server-connection channel to the user ID the channel should run with.
4. Using channel authentication records on channels that do not use SSL or TLS:
   a. Prevent any IP address from opening channels, by using an address-mapping channel authentication record with ADDRESS(*) and USERSRC(NOACCESS).
   b. Allow specific IP addresses to open channels, by using address-mapping channel authentication records for those addresses with USERSRC(CHANNEL).
5. Using a security exit:
   a. Write a security exit to authorize connections based on any property you choose, for example, the originating IP address.
6. It is also possible to use channel authentication records with a security exit, or to use all three methods, if your particular circumstances require it.

Blocking specific IP addresses:

You can prevent a specific channel accepting an inbound connection from an IP address, or prevent the whole queue manager from allowing access from an IP address, by using a channel authentication record.

Before you begin

Enable channel authentication records by running the following command:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

To disallow specific channels from accepting an inbound connection and ensure that connections are only accepted when using the correct channel name, one type of rule can be used to block IP addresses. To disallow an IP address access to the whole queue manager, you would normally use a firewall to permanently block it. However, another type of rule can be used to allow you to block a few addresses temporarily, for example while you are waiting for the firewall to be updated.

Procedure

To block IP addresses from using a specific channel, set a channel authentication record by using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**.

```
SET CHLAUTH(generic-channel-name) TYPE(ADDRESSMAP) ADDRESS(generic-ip-address)
USERSRC(NOACCESS)
```

There are three parts to the command:

SET CHLAUTH (*generic-channel-name*)
: You use this part of the command to control whether you want to block a connection for the entire queue manager, single channel or range of channels. What you put in here determines which areas are covered.

  For example:

  - `SET CHLAUTH('*')` - blocks every channel on a queue manager, that is, the entire queue manager
  - SET CHLAUTH('SYSTEM.*') - blocks every channel that begins with SYSTEM.
  - SET CHLAUTH('SYSTEM.DEF.SVRCONN') - blocks the channel SYSTEM.DEF.SVRCONN

Type of CHLAUTH rule
: Use this part of the command to specify the type of command and determines whether you want to supply a single address or list of addresses.

  For example:

  - `TYPE(ADDRESSMAP)` - Use ADDRESSMAP if you want to supply a single address or wild card address. For example, `ADDRESS('192.168.*')` blocks any connections coming from an IP address starting in `192.168`.

    For more information about filtering IP addresses with patterns, see Generic IP addresses.
  - `TYPE(BLOCKADDR)` - Use BLOCKADDR if you want to supply a list of address to block.

Additional parameters
: These parameters are dependent upon the type of rule you used in the second part of the command:

  - For `TYPE(ADDRESSMAP)` you use ADDRESS
  - For `TYPE(BLOCKADDR)` you use ADDRLIST

Related information:
SET CHLAUTH

Temporarily blocking specific IP addresses if the queue manager is not running:

You might want to block particular IP addresses, or ranges of addresses, when the queue manager is not running and you cannot therefore issue MQSC commands. You can temporarily block IP addresses on an exceptional basis by modifying the `blockaddr.ini` file.

About this task

The `blockaddr.ini` file contains a copy of the BLOCKADDR definitions that are used by the queue manager. This file is read by the listener if the listener is started before the queue manager. In these circumstances, the listener uses any values that you have manually added to the `blockaddr.ini` file.

However, be aware that when the queue manager is started, it writes the set of BLOCKADDR definitions to the `blockaddr.ini` file, over-writing any manual editing you might have done. Similarly, every time you add or delete a BLOCKADDR definition by using the **SET CHLAUTH** command, the `blockaddr.ini` file is updated. You can therefore make permanent changes to the BLOCKADDR definitions only by using the **SET CHLAUTH** command when the queue manager is running.

Procedure
1. Open the `blockaddr.ini` file in a text editor. The file is located in the data directory of the queue manager.
2. Add IP addresses as simple keyword-value pairs, where the keyword is `Addr`. For information about filtering IP addresses with patterns, see Generic IP addresses. For example:
   ```
   Addr = 192.0.2.0
   Addr = 192.0.*
   Addr = 192.0.2.1-8
   ```

Related tasks:
"Blocking specific IP addresses" on page 762
You can prevent a specific channel accepting an inbound connection from an IP address, or prevent the whole queue manager from allowing access from an IP address, by using a channel authentication record.

Related information:
SET CHLAUTH

Blocking specific user IDs:

You can prevent specific users from using a channel by specifying user IDs that, if asserted, cause the channel to end. Do this by setting a channel authentication record.

Before you begin

Ensure that channel authentication records are enabled as follows:
```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:
```
SET CHLAUTH(' generic-channel-name ') TYPE(BLOCKUSER) USERLIST(userID1, userID2)
```
generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

The user list provided on a TYPE(BLOCKUSER) only applies to SVRCONN channels and not queue manager to queue manager channels.

userID1 and *userID2* are each the ID of a user that is to be prevented from using the channel. You can also specify the special value *MQADMIN to refer to privileged administrative users. For more information about privileged users, see "Privileged users" on page 718. For more information about *MQADMIN, see SET CHLAUTH.

Related information:

SET CHLAUTH

Mapping a remote queue manager to an MCAUSER user ID:

You can use a channel authentication record to set the MCAUSER attribute of a channel, according to the queue manager from which the channel is connecting.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Optionally, you can restrict the IP addresses to which the rule applies.

Note that this technique does not apply to server-connection channels. If you specify the name of a server-connection channel in the following commands, it has no effect.

Procedure

- Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

  ```
  SET CHLAUTH(' generic-channel-name ') TYPE (QMGRMAP) QMNAME(generic-partner-qmgr-name
  ) USERSRC(MAP) MCAUSER(user)
  ```

  generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

  generic-partner-qmgr-name is either the name of the queue manager, or a pattern including the asterisk (*) symbol as a wildcard that matches the queue manager name.

  user is the user ID to be used for all connections from the specified queue manager.
- To restrict this command to certain IP addresses, include the **ADDRESS** parameter, as follows:

  ```
  SET CHLAUTH(' generic-channel-name ') TYPE (QMGRMAP) QMNAME(generic-partner-qmgr-name
  ) USERSRC(MAP) MCAUSER(user) ADDRESS(generic-ip-address)
  ```

  generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

  generic-ip-address is either a single address, or a pattern including the asterisk (*) symbol as a wildcard or the hyphen (-) to indicate a range, that matches the address. For more information about generic IP addresses, see Generic IP addresses.

Related information:
SET CHLAUTH

Mapping a client asserted user ID to an MCAUSER user ID:

You can use a channel authentication record to change the MCAUSER attribute of a server-connection channel, according to the original user ID received from a client.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Note that this technique applies only to server-connection channels. It has no effect on other channel types.

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record** . For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE (USERMAP) CLNTUSER(client-user-name) USERSRC(MAP) MCAUSER(user)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

client-user-name is the user ID asserted by the client.

user is the user ID to be used instead of the client user name.

Related information:
SET CHLAUTH

Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID:

You can use a channel authentication record to set the MCAUSER attribute of a channel, according to the Distinguished Name (DN) received.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH('generic-channel-name') TYPE (SSLPEERMAP)
SSLPEER(generic-ssl-peer-name) SSLCERTI(generic-issuer-name)
USERSRC(MAP) MCAUSER(user)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

generic-ssl-peer-name is a string following the standard IBM MQ rules for SSLPEER values. See IBM MQ rules for SSLPEER values.

user is the user ID to be used for all connections using the specified DN.

generic-issuer-name refers to the Issuer DN of the certificate to match. This parameter is optional but you should use it, to avoid spuriously matching the wrong certificate, if multiple certificate authorities are in use.

Related information:

SET CHLAUTH

Blocking access from a remote queue manager:

You can use a channel authentication record to prevent a remote queue manager from starting channels.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Note that this technique does not apply to server-connection channels. If you specify the name of a server-connection channel in the following command, it has no effect.

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE(QMGRMAP) QMNAME(' generic-partner-qmgr-name ') USERSRC(NOACCESS)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

generic-partner-qmgr-name is either the name of the queue manager, or a pattern including the asterisk (*) symbol as a wildcard that matches the queue manager name.

Related information:

SET CHLAUTH

Blocking access for a client asserted user ID:

You can use a channel authentication record to prevent a client asserted user ID from starting channels.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Note that this technique applies only to server-connection channels. It has no effect on other channel types.

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE(USERMAP) CLNTUSER(' client-user-name ') USERSRC(NOACCESS)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

client-user-name is the user ID asserted by the client.

Related information:

SET CHLAUTH

Blocking access for an SSL Distinguished Name:

You can use a channel authentication record to prevent an SSL Distinguished Name (DN) from starting channels.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH('generic-channel-name') TYPE(SSLPEERMAP)
SSLPEER('generic-ssl-peer-name') SSLCERTI(generic-issuer-name)
USERSRC(NOACCESS)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

generic-ssl-peer-name is a string following the standard IBM MQ rules for SSLPEER values. See IBM MQ rules for SSLPEER values.

generic-issuer-name refers to the Issuer DN of the certificate to match. This parameter is optional but you should use it, to avoid spuriously matching the wrong certificate, if multiple certificate authorities are in use.

Related information:

SET CHLAUTH

Mapping an IP address to an MCAUSER user ID:

You can use a channel authentication record to set the MCAUSER attribute of a channel, according to the IP address from which the connection is received.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE(ADDRESSMAP) ADDRESS(' generic-ip-address ') USERSRC(MAP) MCAUSER(user)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

user is the user ID to be used for all connections using the specified DN.

generic-ip-address is either the address from which the connection is being made, or a pattern including the asterisk (*) as a wildcard or the hyphen (-) to indicate a range, that matches the address.

Related information:

SET CHLAUTH

Disabling remote access to the queue manager

If you do not want client applications to connect to your queue manager, disable remote access to it.

About this task

Prevent client applications connecting to the queue manager in one of the following ways:

Procedure

- Delete all server-connection channels using the MQSC command **`DELETE CHANNEL`**.
- Set the message channel agent user identifier (MCAUSER) of the channel to a user ID with no access rights, using the MQSC command **`ALTER CHANNEL`**.

Setting up connection security

Grant the authority to connect to the queue manager to each user or group of users with a business need to do so.

About this task

To set up connection security, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **`SET AUTHREC`** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -t qmgr -g GroupName +connect
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ('QMgrName') OBJTYPE(*MQM) USER('GroupName') AUT(*CONNECT)
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
  PERMIT QMgrName.BATCH CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.CICS UACC(NONE)
  PERMIT QMgrName.CICS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.IMS UACC(NONE)
  PERMIT QMgrName.IMS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.CHIN UACC(NONE)
  PERMIT QMgrName.CHIN CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  ```

  These commands give authority to connect for batch, CICS, IMS and the channel initiator (CHIN). If you do not use a particular type of connection, omit the relevant commands. The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Related information:
Connection security profiles for the channel initiator

Controlling user access to queues

You want to control application access to queues. Use this topic to determine what actions to take.

For each true statement in the first column, take the action indicated in the second column.

Statement	Action
The application gets messages from a queue	See “Granting authority to get messages from queues”
The application sets context	See “Granting authority to set context” on page 770
The application passes context	See “Granting authority to pass context” on page 770
The application puts messages on a clustered queue	See “Authorizing putting messages on remote cluster queues” on page 814
The application puts messages on a local queue	See “Granting authority to put messages to a local queue” on page 771
The application puts messages on a model queue	See “Granting authority to put messages to a model queue” on page 772
The application puts messages on a remote queue	See “Granting authority to put messages to a remote cluster queue” on page 773

Granting authority to get messages from queues:

Grant the authority to get messages from a queue or set of queues, to each group of users with a business need for it.

About this task

To grant the authority to get messages from some queues, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +get
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*GET) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
  PERMIT QMgrName. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

GroupName
: The name of the group to be granted access.

Granting authority to set context:

Grant the authority to set context on a message that is being put, to each group of users with a business need for it.

About this task

To grant the authority to set context on some queues, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue one of the following commands:
  - To set identity context only:
    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +setid
    ```
  - To set all context:
    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +setall
    ```

  Note: To use `setid` or `setall` authority, authorizations must be granted on both the appropriate queue object and also on the queue manager object.
- For IBM i, issue one of the following commands:
  - To set identity context only:
    ```
    GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*SETID) MQMNAME(' QMgrName ')
    ```
  - To set all context:
    ```
    GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*SETALL) MQMNAME(' QMgrName ')
    ```
- For z/OS, issue one of the following sets of commands:
  - To set identity context only:
    ```
    RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
    PERMIT QMgrName. ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
    ```
  - To set all context:
    ```
    RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
    PERMIT QMgrName. ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(CONTROL)
    ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Granting authority to pass context:

Grant the authority to pass context from a retrieved message to one that is being put, to each group of users with a business need for it.

About this task

To grant the authority to pass context on some queues, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue one of the following commands:
  - To pass identity context only:
    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +passid
    ```
  - To pass all context:
    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +passall
    ```
- For IBM i, issue one of the following commands:
  - To pass identity context only:
    ```
    GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PASSID) MQMNAME(' QMgrName ')
    ```
  - To pass all context:
    ```
    GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PASSALL) MQMNAME(' QMgrName ')
    ```
- For z/OS, issue the following commands to pass identity context or all context:
  ```
  RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
  PERMIT QMgrName. ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting authority to put messages to a local queue:

Grant the authority to put messages to a local queue or set of queues, to each group of users with a business need for it.

About this task

To grant the authority to put messages to some local queues, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +put
  ```
- For IBM i, issue the following command:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
```

- For z/OS, issue the following commands:

```
RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
PERMIT QMgrName. ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting authority to put messages to a model queue:

Grant the authority to put messages to a model queue or set of model queues, to each group of users with a business need for it.

About this task

Model queues are used to create dynamic queues. You must therefore grant authority to both the model and dynamic queues. To grant these authorities, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following commands:

```
setmqaut -m QMgrName -n ModelQueueName -t queue -g GroupName +put
setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +put
```

- For IBM i, issue the following commands:

```
GRTMQMAUT OBJ(' ModelQueueName ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
```

- For z/OS, issue the following commands:

```
RDEFINE MQQUEUE QMgrName. ModelQueueName UACC(NONE)
PERMIT QMgrName. ModelQueueName CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
PERMIT QMgrName. ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

ModelQueueName
The name of the model queue on which dynamic queues are based.

ObjectProfile
The name of the dynamic queue or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting authority to put messages to a remote cluster queue:

Grant the authority to put messages to a remote cluster queue or set of queues, to each group of users with a business need for it.

About this task

To put a message on a remote cluster queue, you can either put it on a local definition of a remote queue, or a fully qualified remote queue. If you are using a local definition of a remote queue, you need authority to put to the local object: see “Granting authority to put messages to a local queue” on page 771. If you are using a fully qualified remote queue, you need authority to put to the remote queue. Grant this authority using the appropriate commands for your operating system.

The default behavior is to perform access control against the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. Note that this behavior applies, even if you are using multiple transmission queues.

The specific behavior described in this topic applies only when you have configured the **`ClusterQueueAccessControl`** attribute in the `qm.ini` file to be *RQMName*, as described in the Security stanza topic, and restarted the queue manager.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **`SET AUTHREC`** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -t rqmname -n ObjectProfile -g GroupName +put
  ```
  Note that you can use the *rqmname* object for remote cluster queues only.
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJTYPE(*RMTMQMNAME) OBJ('ObjectProfile') USER(GroupName) AUT(*PUT) MQMNAME('QMgrName')
  ```
  Note that you can use the RMTMQMNAME object for remote cluster queues only.
- For z/OS, issue the following commands:
  ```
  RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
  PERMIT QMgrName.QUEUE. ObjectProfile CLASS(MQQUEUE)
  ID(GroupName) ACCESS(UPDATE)
  ```
  Note that you can use the name of the remote queue manager (or queue-sharing group) for remote cluster queues only. The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  : The name of the remote queue manager or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Controlling user access to topics

You need to control the access of applications to topics. Use this topic to determine what actions to take.

For each true statement in the first column, take the action indicated in the second column.

Table 81. Controlling user access to topics

Statement	Action
The application publishes messages to a topic	See "Granting authority to publish messages to a topic"
The application subscribes to a topic	See "Granting authority to subscribe to topics"

Granting authority to publish messages to a topic:

Grant the authority to publish messages to a topic or set of topics, to each group of users with a business need for it.

About this task

To grant the authority to publish messages to some topics, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName +pub
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(*PUB) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQTOPIC QMgrName. ObjectProfile UACC(NONE)
  PERMIT QMgrName. ObjectProfile CLASS(MQTOPIC) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting authority to subscribe to topics:

Grant the authority to subscribe to a topic or set of topics, to each group of users with a business need for it.

About this task

To grant the authority to subscribe to some topics, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName +sub
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(*SUB) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQTOPIC QMgrName.SUBSCRIBE. ObjectProfile UACC(NONE)
  PERMIT QMgrName.SUBSCRIBE. ObjectProfile CLASS(MQTOPIC) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting authority to inquire on a queue manager

Grant the authority to inquire on a queue manager, to each group of users with a business need for it.

About this task

To grant the authority to inquire on a queue manager, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  ```
  setmqaut -m QMgrName -n ObjectProfile -t qmgr -g GroupName +inq
  ```
- For IBM i, issue the following command:
  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*MQM) USER(GroupName) AUT(*INQ) MQMNAME(' QMgrName ')
  ```
- For z/OS, issue the following commands:
  ```
  RDEFINE MQCMDS QMgrName. ObjectProfile UACC(NONE)
  PERMIT QMgrName. ObjectProfile CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  These commands grant access to the specified queue manager. To permit the user to use the MQINQ command, issue the following commands:
  ```
  RDEFINE MQCMDS QMgrName.MQINQ.QMGR UACC(NONE)
  PERMIT QMgrName.MQINQ.QMGR CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

GroupName
: The name of the group to be granted access.

Granting authority to access processes

Grant the authority to access a process or set of processes, to each group of users with a business need for it.

About this task

To grant the authority to access some processes, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
```
setmqaut -m QMgrName -n ObjectProfile -t process -g GroupName +all
```
- For IBM i, issue the following command:
```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*PRC) USER(GroupName) AUT(*ALL) MQMNAME(' QMgrName ')
```
- For z/OS, issue the following commands:
```
RDEFINE MQPROC QMgrName. ObjectProfile UACC(NONE)
PERMIT QMgrName. ObjectProfile CLASS(MQPROC) ID(GroupName) ACCESS(READ)
```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Granting authority to access namelists

Grant the authority to access a namelist or set of namelists, to each group of users with a business need for it.

About this task

To grant the authority to access some namelists, use the appropriate commands for your operating system.

On UNIX, Linux, Windows systems, and IBM i, you can also use the SET AUTHREC command.

Note: On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
```
setmqaut -m QMgrName -n ObjectProfile -t namelist -g GroupName
+all
```
- For IBM i, issue the following command:

```
GRTMQMAUT OBJ('ObjectProfile
') OBJTYPE(*NMLIST) USER(GroupName) AUT(*ALL) MQMNAME('QMgrName')
```

- For z/OS, issue the following commands:

```
RDEFINE MQNLIST QMgrName.ObjectProfile UACC(NONE)
PERMIT QMgrName.ObjectProfile
CLASS(MQNLIST) ID(GroupName) ACCESS(READ)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Authority to administer IBM MQ on UNIX, Linux, and Windows systems

IBM MQ administrators can use all IBM MQ commands and grant authorities for other users. When administrators issue commands to remote queue managers, they must have the required authority on the remote queue manager. Further considerations apply to Windows systems.

IBM MQ administrators have authority to use all IBM MQ commands (including the commands to grant IBM MQ authorities for other users)

To be an IBM MQ administrator, you must be a member of a special group called the *mqm* group

Windows Alternatively, on Windows only, you can be a member of the Administrators group on Windows systems.

The mqm group is created automatically when IBM MQ is installed; add further users to the group to allow them to perform administration. All members of this group have access to all resources. This access can be revoked only by removing a user from the mqm group and issuing the REFRESH SECURITY command.

Administrators can use control commands to administer IBM MQ. One of these control commands is **setmqaut**, which is used to grant authorities to other users to enable them to access or control IBM MQ resources. The PCF commands for managing authority records are available to non-administrators who have been granted dsp and chg authorities on the queue manager. For more information about managing authorities using PCF commands, see Programmable Command Formats.

Administrators must have the required authorities for the MQSC commands to be processed by the remote queue manager. The IBM MQ Explorer issues PCF commands to perform administration tasks. Administrators require no additional authorities to use the IBM MQ Explorer to administer a queue manager on the local system. When the IBM MQ Explorer is used to administer a queue manager on another system, administrators must have the required authorities for the PCF commands to be processed by the remote queue manager.

Attention: From IBM MQ Version 8.0, you do not have to be an administrator to use the control command **runmqsc**, that issues IBM MQ Script (MQSC) commands.

When **runmqsc** is used in indirect mode to send MQSC commands to a remote queue manager, each MQSC command is encapsulated within an Escape PCF command.

For more information about authority checks when PCF and MQSC commands are processed, see the following topics:

- For PCF commands that operate on queue managers, queues, processes, namelists, and authentication information objects, see Authority to work with IBM MQ objects. Refer to this section for the equivalent MQSC commands encapsulated within Escape PCF commands.
- For PCF commands that operate on channels, channel initiators, listeners, and clusters, see Channel security.
- For PCF commands that operate on authority records, see Authority checking for PCF commands
- z/OS For MQSC commands that are processed by the command server on IBM MQ for z/OS, see Command security and command resource security on z/OS.

Additionally, on Windows systems, the SYSTEM account has full access to IBM MQ resources.

On UNIX and Linux platforms, a special user ID of mqm is also created, for use by the product only. It must never be available to non-privileged users. All IBM MQ objects are owned by user ID mqm.

On Windows systems, members of the Administrators group can also administer any queue manager, as can the SYSTEM account. You can also create a domain mqm group on the domain controller that contains all privileged user IDs active within the domain, and add it to the local mqm group. Some commands, for example **`crtmqm`**, manipulate authorities on IBM MQ objects and so need authority to work with these objects (as described in the following sections). Members of the mqm group have authority to work with all objects, but there might be circumstances on Windows systems when authority is denied if you have a local user and a domain-authenticated user with the same name. This is described in "Principals and groups" on page 782.

Windows versions with a User Account Control (UAC) feature restricts the actions users can perform on certain operating system facilities, even if they are members of the Administrators group. If your userid is in the Administrators group but not the mqm group you must use an elevated command prompt to issue IBM MQ admin commands such as **`crtmqm`**, otherwise the error `AMQ7077: You are not authorized to perform the requested operation` is generated. To open an elevated command prompt, right-click the start menu item, or icon, for the command prompt, and select **Run as administrator**.

You do not need to be a member of the mqm group to take the following actions:

- Issue commands from an application program that issues PCF commands, or MQSC commands within an Escape PCF command, unless the commands manipulate channel initiators. (These commands are described in "Protecting channel initiator definitions" on page 487).
- Issue MQI calls from an application program (unless you want to use the fast path bindings on the **MQCONNX** call).
- Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures.
- Use the **dspmq** command to display queue managers.
- Use the **dspmqtrc** command to display IBM MQ formatted trace output.

A 12 character limitation applies to both group and user IDs.

UNIX and Linux platforms generally restrict the length of a user ID to 12 characters. AIX Version 5.3 has raised this limit but IBM MQ continues to observe a 12 character restriction on all UNIX and Linux platforms. If you use a user ID of greater than 12 characters, IBM MQ replaces it with the value `UNKNOWN` . Do not define a user ID with a value of `UNKNOWN` .

Managing the mqm group

Users in the mqm group are granted full administrative privileges over IBM MQ. For this reason, you should not enrol applications and ordinary users in the mqm group. The mqm group should contain the accounts of the IBM MQ administrators only.

These tasks are described in:
- Creating and managing groups on Windows
- Creating and managing groups on HP-UX
- Creating and managing groups on AIX
- Creating and managing groups on Solaris
- Creating and managing groups on Linux

If your domain controller runs on Windows 2000 or Windows 2003, your domain administrator might have to set up a special account for IBM MQ to use. This is described in the Configuring IBM MQ accounts.

Authority to work with IBM MQ objects on UNIX, Linux and Windows systems

All objects are protected by IBM MQ, and principals must be given appropriate authority to access them. Different principals need different access rights to different objects.

Queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects are all accessed from applications that use MQI calls or PCF commands. These resources are all protected by IBM MQ, and applications need to be given permission to access them. The entity making the request might be a user, an application program that issues an MQI call, or an administration program that issues a PCF command. The identifier of the requester is referred to as the *principal*.

Different groups of principals can be granted different types of access authority to the same object. For example, for a specific queue, one group might be allowed to perform both put and get operations; another group might be allowed only to browse the queue (**MQGET** with browse option). Similarly, some groups might have put and get authority to a queue, but not be allowed to alter attributes of the queue or delete it.

Some operations are particularly sensitive and should be limited to privileged users. For example:
- Accessing some special queues, such as transmission queues or the command queue SYSTEM.ADMIN.COMMAND.QUEUE
- Running programs that use full MQI context options
- Creating and deleting application queues

Full access permission to an object is automatically given to the user ID that created the object and to all members of the mqm group (and to the members of the local Administrators group on Windows systems).

Related concepts:

"Authority to administer IBM MQ on UNIX, Linux, and Windows systems" on page 777
IBM MQ administrators can use all IBM MQ commands and grant authorities for other users. When administrators issue commands to remote queue managers, they must have the required authority on the remote queue manager. Further considerations apply to Windows systems.

When security checks are made on UNIX, Linux and Windows systems

Security checks are typically made on connecting to a queue manager, opening or closing objects, and putting or getting messages.

The security checks made for a typical application are as follows:

Connecting to the queue manager (MQCONN or MQCONNX calls)

This is the first time that the application is associated with a particular queue manager. The queue manager interrogates the operating environment to discover the user ID associated with the application. IBM MQ then verifies that the user ID is authorized to connect to the queue manager and retains the user ID for future checks.

Users do not have to sign on to IBM MQ; IBM MQ assumes that users have signed on to the underlying operating system and have been authenticated by that.

Opening the object (MQOPEN or MQPUT1 calls)

IBM MQ objects are accessed by opening the object and issuing commands against it. All resource checks are performed when the object is opened, rather than when it is actually accessed. This means that the **MQOPEN** request must specify the type of access required (for example, whether the user wants only to browse the object or perform an update like putting messages onto a queue).

IBM MQ checks the resource that is named in the **MQOPEN** request. For an alias or remote queue object, the authorization used is that of the object itself, not the queue to which the alias or remote queue resolves. This means that the user does not need permission to access it. Limit the authority to create queues to privileged users. If you do not, users might bypass the normal access control simply by creating an alias. If a remote queue is referred to explicitly with both the queue and queue manager names, the transmission queue associated with the remote queue manager is checked.

The authority to a dynamic queue is based on that of the model queue from which it is derived, but is not necessarily the same. This is described in Note 1 on page 521.

The user ID used by the queue manager for access checks is the user ID obtained from the operating environment of the application connected to the queue manager. A suitably authorized application can issue an **MQOPEN** call specifying an alternative user ID; access control checks are then made on the alternative user ID. This does not change the user ID associated with the application, only that used for access control checks.

Putting and getting messages (MQPUT or MQGET calls)

No access control checks are performed.

Closing the object (MQCLOSE)

No access control checks are performed, unless the **MQCLOSE** results in a dynamic queue being deleted. In this case, there is a check that the user ID is authorized to delete the queue.

Subscribing to a topic (MQSUB)

When an application subscribes to a topic, it specifies the type of operation that it needs to perform. It is either creating a new subscription, altering an existing subscription, or resuming an existing subscription without changing it. For each type of operation, the queue manager checks that the user ID that is associated with the application has the authority to perform the operation.

When an application subscribes to a topic, the authority checks are performed against the topic objects that are found in the topic tree at or above the point in the topic tree at which the application subscribed. The authority checks might involve checks on more than one topic object.

The user ID that the queue manager uses for the authority checks is the user ID obtained from the operating system when the application connects to the queue manager.

The queue manager performs authority checks on subscriber queues but not on managed queues.

How access control is implemented by IBM MQ on UNIX, Linux and Windows systems

IBM MQ uses the security services provided by the underlying operating system, using the object authority manager. IBM MQ supplies commands to create and maintain access control lists.

An access control interface called the Authorization Service Interface is part of IBM MQ. IBM MQ supplies an implementation of an access control manager (conforming to the Authorization Service Interface) known as the *object authority manager (OAM)*. This is automatically installed and enabled for each queue manager you create, unless you specify otherwise (as described in "Preventing security access checks on Windows, UNIX and Linux systems" on page 744). The OAM can be replaced by any user or vendor written component that conforms to the Authorization Service Interface.

The OAM exploits the security features of the underlying operating system, using operating system user and group IDs. Users can access IBM MQ objects only if they have the correct authority. "Controlling access to objects by using the OAM on UNIX, Linux and Windows systems" on page 735 describes how to grant and revoke this authority.

The OAM maintains an access control list (ACL) for each resource that it controls. Authorization data is stored on a local queue called SYSTEM.AUTH.DATA.QUEUE. Access to this queue is restricted to users in the mqm group, and additionally on Windows, to users in the Administrators group, and users logged in with the SYSTEM ID. User access to the queue cannot be changed.

IBM MQ supplies commands to create and maintain access control lists. For more information on these commands, see "Controlling access to objects by using the OAM on UNIX, Linux and Windows systems" on page 735.

IBM MQ passes the OAM a request containing a principal, a resource name, and an access type. The OAM grants or rejects access based on the ACL that it maintains. IBM MQ follows the decision of the OAM; if the OAM cannot make a decision, IBM MQ does not allow access.

Identifying the user ID on Windows, UNIX and Linux systems

The object authority manager identifies the principal that is requesting access to a resource. The user ID used as the principal varies according to context.

The object authority manager (OAM) must be able to identify who is requesting access to a particular resource. IBM MQ uses the term *principal* to refer to this identifier. The principal is established when the application first connects to the queue manager; it is determined by the queue manager from the user ID associated with the connecting application. (If the application issues XA calls without connecting to the queue manager, then the user ID associated with the application that issues the xa_open call is used for authority checks by the queue manager.)

On UNIX and Linux systems, the authorization routines checks either the real (logged-in) user ID, or the effective user ID associated with the application. The user ID checked can be dependent on the bind type, for details see Installable services.

IBM MQ propagates the user ID received from the system in the message header (MQMD structure) of each message as identification of the user. This identifier is part of the message context information and is described in "Context authority on UNIX, Linux and Windows systems" on page 784. Applications cannot alter this information unless they have been authorized to change context information.

Principals and groups

Principals can belong to groups. By granting resource access to groups rather than to individuals, you can reduce the amount of administration required. Access Control Lists (ACLs) are based on both groups and user IDs.

For example, you might define a group consisting of users who want to run a particular application. Other users can be given access to all the resources they require by adding their user ID to the appropriate group.

This process of defining and managing groups is described for particular platforms:
- Creating and managing groups on Windows
- Creating and managing groups on HP-UX
- Creating and managing groups on AIX
- Creating and managing groups on Solaris
- Creating and managing groups on Linux

A principal can belong to more than one group (its group set). It has the aggregate of all the authorities granted to each group in its group set. These authorities are cached, so any changes you make to the group membership of the principal are not recognized until the queue manager is restarted, unless you issue the MQSC command REFRESH SECURITY (or its PCF equivalent).

UNIX and Linux systems

ACLs are based on both user IDs and groups and you can use either for authorization.

With Version 8.0, you can use the *user-based model* for authorization, and this allows you to use both users and groups. However, when you specify a user in the setmqaut command, the new permissions apply to that user alone, and not any groups to which that user belongs.

See Installable services for more information.

When you are using the *group-based model* for authorization, the primary group to which the user ID belongs is included in the ACL.

The individual user ID is not included and authority is granted to all members of that group. Because of this, be aware that you can inadvertently change the authority of a principal by changing the authority of another principal in the same group.

All users are nominally assigned to the default user group `nobody` and by default, no authorizations are given to this group. You can change the authorization in the `nobody` group to grant access to IBM MQ resources to users without specific authorizations.

Do not define a user ID with the value `UNKNOWN`. The value `UNKNOWN` is used when a user ID is too long, so arbitrary user IDs would use the access authorities of UNKNOWN.

User IDs can contain up to 12 characters and group names up to 12 characters.

Windows systems

ACLs are based on both user IDs and groups. Checks are the same as for UNIX systems. You can have different users on different domains with the same user ID. IBM MQ permits user IDs to be qualified by a domain name so that these users can be given different levels of access.

The group name can optionally include a domain name, specified in the following formats:

`GroupName@domain` *`domain_name\group_name`*

Global groups are checked by the OAM in two cases only:
1. The queue manager security stanza includes the setting: `GroupModel=GlobalGroups`. See Security.
2. The queue manager is using an alternative security access group. See **`crtmqm`** .

User IDs can contain up to 20 characters, domain names up to 15 characters, and group names up to 64 characters.

The OAM first checks the local security database, then the database of the primary domain, and finally the database of any trusted domains. The first user ID encountered is used by the OAM for checking. Each of these user IDs might have different group memberships on a particular computer.

Some control commands (for example, **crtmqm**) change authorities on IBM MQ objects using the object authority manager (OAM). The OAM searches the security databases in the order given in the preceding paragraph to determine the authority rights for a particular user ID. As a result, the authority determined by the OAM might override the fact that a user ID is a member of the local mqm group. For example, if you issue the **crtmqm** command from a user ID authenticated by a domain controller that has membership of the local mqm group through a global group, the command fails if the system has a local user with the same name who is not in the local mqm group.

Windows security identifiers (SIDs)

IBM MQ on Windows uses the SID where it is available. If a Windows SID is not supplied with an authorization request, IBM MQ identifies the user based on the user name alone, but this might result in the wrong authority being granted.

On Windows systems, the security identifier (SID) is used to supplement the user ID. The SID contains information that identifies the full user account details on the Windows security account manager (SAM) database where the user is defined. When a message is created on IBM MQ for Windows, IBM MQ stores the SID in the message descriptor. When IBM MQ on Windows performs authorization checks, it uses the SID to query the full information from the SAM database. (The SAM database in which the user is defined must be accessible for this query to succeed.)

By default, if a Windows SID is not supplied with an authorization request, IBM MQ identifies the user based on the user name alone. It does this by searching the security databases in the following order:

1. The local security database
2. The security database of the primary domain
3. The security database of trusted domains

If the user name is not unique, incorrect IBM MQ authority might be granted. To prevent this problem, include an SID in each authorization request; the SID is used by IBM MQ to establish user credentials.

To specify that all authorization requests must include an SID, use **regedit** . Set the SecurityPolicy to NTSIDsRequired.

Alternate-user authority on UNIX, Linux and Windows systems

You can specify that a user ID can use the authority of another user when accessing an IBM MQ object. This is called *alternate-user authority*, and you can use it on any IBM MQ object.

Alternate-user authority is essential where a server receives requests from a program and wants to ensure that the program has the required authority for the request. The server might have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example, assume that a server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1. When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message. Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify a different user ID, in this case, USER1. In this example, you can use alternate-user authority to control whether PAYSERV is allowed to specify USER1 as an alternate-user ID when it opens the reply-to queue.

The alternate-user ID is specified on the **`AlternateUserId`** field of the object descriptor.

Context authority on UNIX, Linux and Windows systems

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message. Applications can specify the context data when either an **MQOPEN** or **MQPUT** call is made.

The context information comes in two sections:

Identity section
Who the message came from. It consists of the `UserIdentifier`, `AccountingToken`, and `ApplIdentityData` fields.

Origin section
Where the message came from, and when it was put onto the queue. It consists of the `PutApplType`, `PutApplName`, `PutDate`, `PutTime`, and `ApplOriginData` fields.

Applications can specify the context data when either an **MQOPEN** or **MQPUT** call is made. This data might be generated by the application, passed on from another message, or generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, testing whether the message came from an application running under an authorized user ID.

A server program can use the `UserIdentifier` to determine the user ID of an alternative user. You use context authorization to control whether the user can specify any of the context options on any **MQOPEN** or **MQPUT1** call.

See Controlling context information for information about the context options, and Overview for MQMD for descriptions of the message descriptor fields relating to context.

Implementing access control in security exits

You can implement access control in a security exit by use of the MCAUserIdentifier or the object authority manager.

MCAUserIdentifier

Every instance of a channel that is current has an associated channel definition structure, MQCD. The initial values of the fields in MQCD are determined by the channel definition that is created by an IBM MQ administrator. In particular, the initial value of one of the fields, *MCAUserIdentifier*, is determined by the value of the MCAUSER parameter on the DEFINE CHANNEL command, or by the equivalent to MCAUSER if the channel definition is created in another way.

The MQCD structure is passed to a channel exit program when it is called by an MCA. When a security exit is called by an MCA, the security exit can change the value of *MCAUserIdentifier*, replacing any value that was specified in the channel definition.

On IBM i, UNIX, Linux and Windows systems, unless the value of *MCAUserIdentifier* is blank, the queue manager uses the value of *MCAUserIdentifier* as the user ID for authority checks when an MCA attempts to access the queue manager's resources after it has connected to the queue manager. If the value of *MCAUserIdentifier* is blank, the queue manager uses the default user ID of the MCA instead. This applies to RCVR, RQSTR, CLUSRCVR and SVRCONN channels. For sending MCAs, the default user ID is always used for authority checks, even if the value of *MCAUserIdentifier* is not blank.

On z/OS, the queue manager might use the value of *MCAUserIdentifier* for authority checks, provided it is not blank. For receiving MCAs and server connection MCAs, whether the queue manager uses the value of *MCAUserIdentifier* for authority checks depends on:
- The value of the PUTAUT parameter in the channel definition
- The RACF profile used for the checks
- The access level of the channel initiator address space user ID to the RESLEVEL profile

For sending MCAs, it depends on:
- Whether the sending MCA is a caller or a responder
- The access level of the channel initiator address space user ID to the RESLEVEL profile

The user ID that a security exit stores in *MCAUserIdentifier* can be acquired in various ways. Here are some examples:
- Provided there is no security exit at the client end of an MQI channel, a user ID associated with the IBM MQ client application flows from the client connection MCA to the server connection MCA when the client application issues an MQCONN call. The server connection MCA stores this user ID in the *RemoteUserIdentifier* field in the channel definition structure, MQCD. If the value of *MCAUserIdentifier* is blank at this time, the MCA stores the same user ID in *MCAUserIdentifier*. If the MCA does not store the user ID in *MCAUserIdentifier*, a security exit can do it later by setting *MCAUserIdentifier* to the value of *RemoteUserIdentifier*.

  If the user ID that flows from the client system is entering a new security domain and is not valid on the server system, the security exit can substitute the user ID for one that is valid and store the substituted user ID in *MCAUserIdentifier*.
- The user ID can be sent by the partner security exit in a security message.

  On a message channel, a security exit called by the sending MCA can send the user ID under which the sending MCA is running. A security exit called by the receiving MCA can then store the user ID in *MCAUserIdentifier*. Similarly, on an MQI channel, a security exit at the client end of the channel can send the user ID associated with the IBM MQ MQI client application. A security exit at the server end of the channel can then store the user ID in *MCAUserIdentifier*. As in the previous example, if the user

ID is not valid on the target system, the security exit can substitute the user ID for one that is valid and store the substituted user ID in *MCAUserIdentifier*.

If a digital certificate is received as part of the identification and authentication service, a security exit can map the Distinguished Name in the certificate to a user ID that is valid on the target system. It can then store the user ID in *MCAUserIdentifier*.

- If SSL is used on the channel, the partner's Distinguished Name (DN) is passed to the exit in the SSLPeerNamePtr field of the MQCD, and the DN of the issuer of that certificate is passed to the exit in the SSLRemCertIssNamePtr field of the MQCXP.

For more information about the *MCAUserIdentifier* field, the channel definition structure, MQCD, and the channel exit parameter structure, MQCXP, see Channel-exit calls and data structures. For more information about the user ID that flows from a client system on an MQI channel, see Access control.

Note: Security exit applications constructed prior to the release of IBM MQ v7.1 may require updating. For more information see Channel security exit programs.

IBM MQ object authority manager user authentication

On IBM MQ MQI client connections, security exits can be used to modify or create the MQCSP structure used in object authority manager (OAM) user authentication. This is described in Channel-exit programs for messaging channels

Implementing access control in message exits

You might need to use a message exit to substitute one user ID with another.

Consider a client application that sends a message to a server application. The server application can extract the user ID from the *UserIdentifier* field in the message descriptor and, provided it has alternate user authority, ask the queue manager to use this user ID for authority checks when it accesses IBM MQ resources on behalf of the client.

If the PUTAUT parameter is set to CTX (or ALTMCA on z/OS) in the channel definition, the user ID in the *UserIdentifier* field of each incoming message is used for authority checks when the MCA opens the destination queue.

In certain circumstances, when a report message is generated, it is put using the authority of the user ID in the *UserIdentifier* field of the message causing the report. In particular, confirm-on-delivery (COD) reports and expiration reports are always put with this authority.

Because of these situations, it might be necessary to substitute one user ID for another in the *UserIdentifier* field as a message enters a new security domain. This can be done by a message exit at the receiving end of the channel. Alternatively, you can ensure that the user ID in the *UserIdentifier* field of an incoming message is defined in the new security domain.

If an incoming message contains a digital certificate for the user of the application that sent the message, a message exit can validate the certificate and map the Distinguished Name in the certificate to a user ID that is valid on the receiving system. It can then set the *UserIdentifier* field in the message descriptor to this user ID.

If it is necessary for a message exit to change the value of the *UserIdentifier* field in an incoming message, it might be appropriate for the message exit to authenticate the sender of the message at the same time. For more details, see "Identity mapping in message exits" on page 720.

Implementing access control in the API exit and API-crossing exit

An API or API-crossing exit can provide access controls to supplement those provided by IBM MQ. In particular, the exit can provide access control at the message level. The exit can ensure that an application puts on a queue, or gets from a queue, only those messages that satisfy certain criteria.

Consider the following examples:

- A message contains information about an order. When an application attempts to put a message on a queue, an API or API-crossing exit can check that the total value of the order is less than some prescribed limit.
- Messages arrive on a destination queue from remote queue managers. When an application attempts to get a message from the queue, an API or API-crossing exit can check that the sender of the message is authorized to send a message to the queue.

Overview of LDAP authorization

V 8.0.0.2

An overview of how you use LDAP authorization to remove the need for a local user ID.

Commands that handle authorization configuration, such as setmqaut and DISPLAY AUTHREC, can now process Distinguished Names. Previously, users were authenticated by comparing their credentials with the maximum available characters that exist for users and groups on the local operating system.

Attention: If you have run the DEFINE AUTHINFO command, you must restart the queue manager. If you do not restart the queue manager, the setmqaut command does not return the correct result.

If a user provides a user ID, rather than a Distinguished Name, the user ID is processed. For example, when there is an incoming message on a channel with PUTAUT(CTX), the characters in the user ID are mapped to an LDAP Distinguished Name, and the appropriate authorization checks are made.

Other commands such as DISPLAY CONN, continue to work with and show the actual value for the user ID, even though that user ID might not actually exist on the local OS.

When LDAP authorization is in place, the queue manager always uses the user model of security on UNIX platforms, regardless of the **SecurityPolicy** attribute in the `qm.ini` file. So, setting permissions for an individual user affects only that user, and not anyone else who belongs to any of that user's groups.

As with the OS model, a user still has the combined authority that has been assigned to both the individual and to all of the groups (if any) to which the user belongs.

For example, assume that the following records have been defined in an LDAP repository.

- In the **inetOrgPerson** class:

```
dn="cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry"
        email=JohnDoe1@yourcompany.com [longer than 12 characters]
        shortu=jodoe
        Phone=1234567
```

- In the **groupOfNames** class:

```
dn="cn=Application Group A, ou=groups, o=yourcompany, c=yourcountry"
        longname=ApplicationGroupA [longer than 12 characters]
        members="cn=JaneDoe, ou=users, o=yourcompany, c=yourcountry",
                "cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry"
```

For authentication purposes, a queue manager using this LDAP server must have been defined so that its CONNAUTH value points at an AUTHINFO object of type IDPWLDAP, and whose relevant name-resolution attributes are probably set as follows:

```
USRFIELD(email) SHORTUSR(shortu)
BASEDNU(ou=users,o=yourcompany,c=yourcountry) CLASSUSR(inetOrgPerson)
```

Given this configuration for authentication, an application can fill in the CSPUserID field, used within the MQCNO call, with either of the following sets of values:

```
" cn=JohnDoe ", " JohnDoe1@yourcompany.com ", " email=JohnDoe1@yourcompany.com "
```

or

```
" cn=JohnDoe, ou=users, o=ibm, c=uk ", " shortu=jodoe "
```

In either case, the system can use the supplied values to authenticate the OS context of " `jodoe`".

Enabling LDAP authorization on the supported platforms

LDAP authorization is available on the following platforms:
- UNIX
- IBM i

Important:

To enable configuration of the new authorization model, you must firstly change the command level of the queue manager to be higher than the IBM MQ Version 8.0.0.0 general availability level of *800*.

You have to do this explicitly. The change is not done automatically by installation of the fix pack. For example:

```
strmqm -e CMDLEVEL=802 <qmname>
strmqm <qmname>
```

If you set **`CMDLEVEL`** to *802*, you can rollback the fix pack, provided that you have not used the new function.

The queue manager command level controls whether or not configuration changes can be made that affect authorization. After this change you can see and modify the new attributes on the AUTHINFO object.

If you attempt to access the new AUTHINFO attributes from an upgraded code level, without also updating the command level, you receive an error message about needing to change the command level.

Note that the command level has no direct relationship to the Version, Release, Modification, Fix pack level.

Setting authorizations

V 8.0.0.2

How you use the short name or **`USRFIELD`** to set authorizations.

The approach of working with multiple formats, described in "Overview of LDAP authorization" on page 789, continues into the authorization commands, with a further extension that either the `shortname` or the USRFIELD can be used in an unadorned fashion.

The character string specifies a particular attribute in the LDAP record when naming users (principals) for authorization.

Important: The character string must not contain the = character, because this character cannot be used in an operating system user ID.

If you pass a principal name to the OAM for authorization that is potentially a `shortname`, the character string must fit into 12 characters. The mapping algorithm first tries to resolve it to a DN using the SHORTUSR attribute in its LDAP query.

If that fails with an UNKNOWN_ENTITY error, or if the given string cannot possibly be a `shortname`, a further attempt is made using the USRFIELD attribute to construct the LDAP query.

Attention: If you have run the DEFINE AUTHINFO command, you must restart the queue manager. If you do not restart the queue manager, the setmqaut command does not return the correct result.

For processing user authorizations, using the setmqaut command, the following are all equivalent.

Table 82. User authorization settings

Command	Note
`setmqaut -m QM -t qmgr -p jodoe +connect`	This is a flat, unqualified name, resolved through SHORTUSR.
`setmqaut -m QM -t qmgr -pJohnDoe1@yourcompany.com +connect`	Also a flat, unqualified name, resolving via USRFIELD to the same entity.
`setmqaut -m QM -t qmgr -p email=JohnDoe1@yourcompany.com +connect`	Using a named attribute.
`setmqaut -m QM -t qmgr -p "phone=1234567" +connect`	Using another named attribute which does not have to be any of those configured on the AUTHINFO object.

You can use the SET AUTHREC MQSC command as an alternative to the **setmqaut** command:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL('JohnDoe1@yourcompany.com') AUTHADD(connect)
```

or the Set Authority Record (MQCMD_SET_AUTH_REC) PCF command with the MQCACF_PRINCIPAL_ENTITY_NAMES element containing the string:

```
"cn=JohnDoe,ou=users,o=yourcompany,c=yourcountry"
```

When processing groups, there is no ambiguity about `shortname` processing, as there is no requirement to fit any form of a group name into 12-characters. Therefore, there is no equivalent of the SHORTUSR attribute for groups.

That means that the syntax examples described in Table 83 are valid, assuming that you have configured the AUTHINFO object with the extended attributes, and set to:

```
GRPFIELD(longname)
BASEDNG(ou=groups,o=yourcompany,c=yourcountry ) CLASSGRP(groupOfNames)
```

Table 83. Group authorization settings

Command	Note
`setmqaut -m QM -t qmgr -g ApplicationGroupA +connect`	Using GRPFIELD to resolve
`setmqaut -m QM -t qmgr -g longname=ApplicationGroupA +connect`	Naming a single attribute
`setmqaut -m QM -t qmgr -g "cn=Application Group A,ou=groups,o=yourcompany,c=yourcountry" +connect`	Using the full DN

You can use the SET AUTHREC MQSC command as an alternative to the preceding **setmqaut** command:

```
SET AUTHREC OBJTYPE(QMGR) GROUP('ApplicationGroupA')
      AUTHADD(connect)
```

or the Set Authority Record (MQCMD_SET_AUTH_REC) PCF command with the MQCACF_GROUP_ENTITY_NAMES element containing the string:

```
"ApplicationGroupA"
```

Important:

Whichever format you use to refer to a name, whether for user or group, it must be possible to derive a unique DN.

So, for example, you must not have two distinct records that both have "`shortu=jodoe`".

If a single unique DN cannot be determined, the OAM returns MQRC_UNKNOWN_ENTITY.

Displaying authorizations

V 8.0.0.2 V 8.0.0.2

Various methods of displaying authorization of users or groups.

dspmqaut command

The simplest method for displaying the authorizations available for a user or group is to use the dspmqaut command.

You can use a query on any of the syntax variations for identifying a user or group. Note that the command output repeats the identity in the format given on the command line. The output does not report on the full resolved DN.

For example:

```
dspmqaut -m QM -t qmgr -p johndoe
Entity johndoe has the following authorizations for object QM:
 connect
```

or

```
dspmqaut -m QM -t qmgr -p email=JohnDoe1@yourcompany.com
Entity email=JohnDoe1@yourcompany.com has the following authorizations for object QM:
 connect
```

dmpmqaut and dmpmqcfg commands

The dmpmqaut command, and its MQSC or PCF equivalents, can specify the principal or group in any of the supported formats, like the **setmqaut** tables described in “Setting authorizations” on page 790. However, unlike **dspmqaut**, the **dmpmqaut** command always reports the full DN.

```
dmpmqaut -m QM -t qmgr -p jodoe
------------------------------------
profile: self
object type:qmgr
entity:cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry
entity type: principal
authority: connect
```

Similarly, the dmpmqcfg command, which does not have any filtering on the selected records, always shows the full DN in a format that can be replayed later.

```
dmpmqcfg -m QM -x authrec
------------------------------------
SET AUTHREC PROFILE(SELF) +
 PRINCIPAL('cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry') +
 OBJTYPE(QMGR)
 AUTHADD(CONNECT)
```

LDAP authorization - other changes

V 8.0.0.2 V 8.0.0.2

A brief description of changes to the Message Queue Interface (MQI) and other MQSC and PCF commands that you need to be aware of when using LDAP authorization from IBM MQ Version 8.0.0, Fix Pack 2.

ADOPTCTX

There is no requirement for applications to provide authentication information, or for the ADOPTCTX attribute to be set to YES.

If an application does not explicitly authenticate, or if **ADOPTCTX** is set to NO for the active CONNAUTH object, the identity context associated with the application is taken from the operating system user ID.

When authorizations need to be applied, that context is mapped to an LDAP identity using the same rules as for the setmqaut commands.

Input parameters to MQI calls

MQOPEN, MQPUT1, and MQSUB have structures that allow an alternative user ID to be specified.

If those fields are used, the 12-character user ID is mapped to a DN using the same rules as on the **setmqaut**, **dmpmqaut**, and **dspmqaut** commands.

MQPUT and MQPUT1 also allow suitably authorized programs to set the MQMD UserIdentifier field. The value of this field is not policed during the PUT process, and can be set to any value.

As usual, however, the **UserIdentifier** value can be used for authorization at later stages of the message processing, for example when PUTAUT(CTX) is defined on a receiving channel.

At that point, the identifier will be checked for authorization using the configuration of that receiving queue manager – which can be LDAP or OS-based.

Output parameters to MQI calls

Wherever a user ID is provided to a program in an MQI structure, it is the 12-character short name version associated with the connection.

For example, the **MQAXC.UserId** value for API Exits is the short name returned from the LDAP mapping.

Other administrative MQSC and PCF commands

Commands that show user information in object status such as DISPLAY CONN USERID return the 12-character short name associated with the context. The full DN is not shown.

Commands that allow assertion of identities, such as the CHLAUTH mapping rules or MCAUSER values for channels, can take values up to the maximum length defined for those attributes (currently 64 characters).

There is no change to the syntax. When authorization is required for that identity, it is internally mapped to a DN using the same rules as for the **setmqaut**, **dmpmqaut**, and **dspmqaut** commands.

This means that the MCAUSER value on a channel definition might not display as the same string as DISPLAY CHSTATUS but they do refer to the same identity.

For example:

```
DEFINE CHL(SV1) CHLTYPE(SVRCONN) MCAUSER('cn=JohnDoe')
DEFINE CHL(SV2) CHLTYPE(SVRCONN) MCAUSER('jodoe')
DEFINE CHL(SV3) CHLTYPE(SVRCONN) MCAUSER('JohnDoe1@yourcompany.com')
```

Then DISPLAY CHSTATUS(*) ALL shows the SHORTUSR value, *MCAUSER(jodoe)* for all connections.

Switching between OS and LDAP authorization models

V 8.0.0.2 V 8.0.0.2

How you switch between the different authorization methods on different platforms.

The CONNAUTH attribute of the queue manager points at an AUTHINFO object. When the object is of type IDPWLDAP, an LDAP repository is used for authentication.

You can now apply an authorization method to that same object, which allows you to continue with OS-based authorization, or to work with LDAP authorization

UNIX platforms and IBM i

The queue manager can be switched at any time between OS and LDAP models. You can change the configuration and make that configuration active by using the REFRESH SECURITY TYPE (CONNAUTH) command.

For example, if this object has already been configured with the connection information for authentication:

```
ALTER AUTHINFO(MYLDAP) AUTHTYPE(IDPWLDAP) +
        AUTHORMD(SEARCHGRP) +
        BASEDNG('ou=groups,o=ibm,c=uk') +
        ⌂other attributes>
ALTER QMGR CONNAUTH(MYLDAP)
REFRESH SECURITY
```

Processing rules

When switching from OS to LDAP authorization, any existing OS authority rules that have been set, become inactive and invisible.

Commands such as **dmpmqaut** do not display those OS rules. Similarly, when switching back from LDAP to OS, any defined LDAP authorizations become inactive and invisible, restoring the original OS rules.

If you want to back up the definitions of a queue manager for any reason, using the **dmpmqcfg** command, then that backup will contain only the rules that are defined for the authorization method in effect at the time of the back up.

LDAP administration

V 8.0.0.2 V 8.0.0.2

An overview of how each platform administers LDAP.

When using LDAP authorization, membership of the `mqm` group (or equivalent) in the operating system is not that important. Being a member of that group only controls whether certain command-line commands can be processed.

In particular, you must be in that group to issue the strmqm and endmqm commands.

Once the queue manager is running, there are now limits on the fully-privileged account. Apart from the user ID of the person who issues the **`strmqm`** command, other users belonging to the OS `mqm` (or equivalent) group do not get special privileges.

Authorizations of other users are based on which LDAP groups they belong to. An unqualified use of the `mqm` group name in commands such as **`setmqaut`** is not allowed to map to any LDAP group.

UNIX platforms

Once the queue manager is running, the only automatically fully-privileged account is the real user who started the queue manager.

The `mqm` ID still exists and is used as the owner of OS resources, such as files, because `mqm` is the effective ID under which the queue manager is running. However, the `mqm` user will not automatically be able to do administrative tasks controlled by the OAM.

IBM i

On IBM i, the automatically-privileged accounts are the one that starts the queue manager and the QMQM ID.

You need both IDs, because the user ID that starts the queue manager is required only to start the system. Once running, the queue manager processes have QMQM authority only.

Sample script

As it is useful to have a group able to do full administration on a queue manager, a sample script is shipped on UNIX platforms as:

```
<install dir>/samp/bin/amqauthg.sh
```

This sample takes two parameters:
- A queue manager name
- An LDAP group name

The sample processes setmqaut commands, granting full authority for all objects. This is the same script that is generated by theMQ Explorer OAM Wizard for administrative roles. For example, the code starts:

```
setmqaut -t q -m <qmgr> -n "**" +alladm +allmqi -g
      <groupname>
```

Confidentiality of messages

To maintain confidentiality, encrypt your messages. There are various methods of encrypting messages in IBM MQ depending on your needs.

Your choice of CipherSpec determines what level of confidentiality you have.

If you need application-level, end-to-end data protection for your point to point messaging infrastructure, you can use IBM MQ Advanced Message Security to encrypt the messages, or write your own API exit or API-crossing exit.

If you need to encrypt messages only while they are being transported through a channel, because you have adequate security on your queue managers, you can use SSL or TLS, or you can write your own security exit, message exit, or send and receive exit programs.

For more information about IBM MQ Advanced Message Security, see “IBM MQ Advanced Message Security” on page 475. The use of SSL and TLS with IBM MQ is described at “SSL and TLS security protocols in IBM MQ” on page 397. The use of exit programs in message encryption is described at “Implementing confidentiality in user exit programs” on page 806.

Related information:

Connecting two queue managers using SSL or TLS

Connecting a client to a queue manager securely

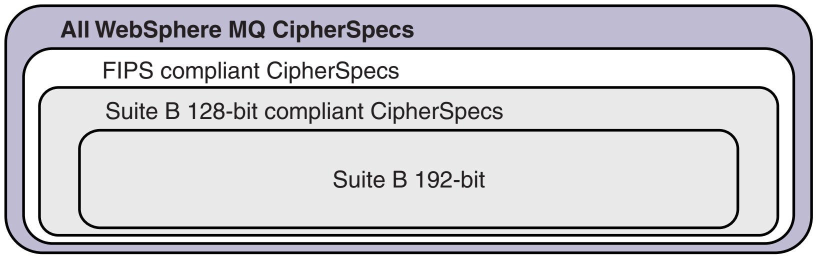
Enabling CipherSpecs

Enable a CipherSpec by using the **SSLCIPH** parameter in either the **DEFINE CHANNEL** MQSC command or the **ALTER CHANNEL** MQSC command.

Some of the CipherSpecs that you can use with IBM MQ are FIPS compliant. Some of the FIPS compliant CipherSpecs are also Suite B compliant although others, such as `TLS_RSA_WITH_3DES_EDE_CBC_SHA` (deprecated), are not.

All Suite B compliant CipherSpecs are also FIPS compliant. All Suite B compliant CipherSpecs fall into two groups: 128 bit (for example, `ECDHE_ECDSA_AES_128_GCM_SHA256`) and 192 bit (for example, `ECDHE_ECDSA_AES_256_GCM_SHA384`),

The following diagram illustrates the relationship between these subsets:


All WebSphere MQ CipherSpecs
FIPS compliant CipherSpecs
Suite B 128-bit compliant CipherSpecs
Suite B 192-bit

V 8.0.0.3 From IBM MQ Version 8.0.0, Fix Pack 3 the number of supported CipherSpecs has been reduced. See “CipherSpec values supported in IBM MQ” on page 413 for more information on the list of supported CipherSpecs and how you can enable deprecated CipherSpecs.

See “Deprecated CipherSpecs” on page 800 for a list of CipherSpecs that you must re-enable to use with IBM MQ.

Cipher specifications that you can use with the IBM MQ queue manager automatically are listed in the following table. When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake is the size stored in the certificate unless it is determined by the CipherSpec, as noted in the table.

Platform support[1]	**CipherSpec name**	**Protocol used**	**Data integrity**	**Encryption algorithm**	**Encryption bits**	**FIPS**[2]	**Suite B**
Linux Windows UNIX z/OS	`TLS_RSA_WITH_AES_128_CBC_SHA`	TLS 1.0	SHA-1	AES	128	Yes	No
Linux Windows UNIX z/OS	`TLS_RSA_WITH_AES_256_CBC_SHA`[3]	TLS 1.0	SHA-1	AES	256	Yes	No
All	`ECDHE_ECDSA_AES_128_CBC_SHA256`	TLS 1.2	SHA-256	AES	128	Yes	No
All	`ECDHE_ECDSA_AES_256_CBC_SHA384` 3 on page 799	TLS 1.2	SHA-384	AES	256	Yes	No
distributed	`ECDHE_ECDSA_AES_128_GCM_SHA256` [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	128 bit
distributed	`ECDHE_ECDSA_AES_256_GCM_SHA384` [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	192 bit
All	`ECDHE_RSA_AES_128_CBC_SHA256`	TLS 1.2	SHA-256	AES	128	Yes	No
All	`ECDHE_RSA_AES_256_CBC_SHA384` 3 on page 799	TLS 1.2	SHA-384	AES	256	Yes	No
distributed	`ECDHE_RSA_AES_128_GCM_SHA256` [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
distributed	`ECDHE_RSA_AES_256_GCM_SHA384` [3] [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No
5 IBM i	`ECDHE_ECDSA_RC4_128_SHA256`	TLS 1.2	AEAD AES-128 GCM	AES	SHA256	Yes	No
IBM i	`ECDHE_ECDSA_3DES_EDE_CBC_SHA256`	TLS 1.2	AEAD AES-128 GCM	3DES	SHA256	Yes	No
IBM i	`ECDHE_RSA_3DES_EDE_CBC_SHA256`	TLS 1.2	AEAD AES-128 GCM	3DES	SHA256	Yes	No
IBM i	`ECDHE_RSA_RC4_128_SHA256`	TLS 1.2	AEAD AES-128 GCM	RSA	SHA256	Yes	No

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B
IBM i	ECDHE_RSA_NULL_SHA256	TLS 1.2	AEAD AES-128 GCM	RSA	SHA256	Yes	No
IBM i	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	AEAD AES-128 GCM	ECDSA	SHA256	Yes	No
IBM i	ECDHE_ECDSA_AES_256_GCM_SHA3843 [4]	TLS 1.2	AEAD AES-128 GCM	AES	SHA384	Yes	No
Linux Windows UNIX z/OS	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	SHA-256	AES	128	Yes	No
Linux Windows UNIX z/OS	TLS_RSA_WITH_AES_256_CBC_SHA256 3	TLS 1.2	SHA-256	AES	256	Yes	No
distributed	TLS_RSA_WITH_AES_128_GCM_SHA256 [4]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No
distributed	TLS_RSA_WITH_AES_256_GCM_SHA3843 [4]	TLS 1.2	AEAD AES-128 GCM	AES	256	Yes	No

Notes:

1. If no specific platform is noted, the CipherSpec is available on all platforms.
2. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
3. This CipherSpec cannot be used to secure a connection from the MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
4. Following a recommendation by NIST, GCM CipherSpecs now have a restriction which means that after 2⌂22 TLS records are sent, using the same session key, the connection will be terminated with message AMQ9288.

   To prevent this error from happening: avoid using GCM Ciphers, enable secret key reset, or start your IBM MQ queue manager with the environment variable GSK_ENFORCE_GCM_RESTRICTION=GSK_FALSE set.
   Important: The GCM restriction is active, regardless of the FIPS mode being used.
5. The CipherSpecs listed as supported on IBM i apply to Versions 7.2 and 7.3 of IBM i.

Related concepts:

"Digital certificates and CipherSpec compatibility in IBM MQ" on page 419
This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

Related information:

DEFINE CHANNEL

ALTER CHANNEL

Deprecated CipherSpecs

A list of deprecated CipherSpecs that you are able to use with IBM MQ if necessary.

See "CipherSpec values supported in IBM MQ" on page 413 for more information on how you can enable deprecated CipherSpecs.

Deprecated CipherSpecs that you can use with IBM MQ TLS and SSL support are listed in the following table.

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B	Update when deprecated
IBM i	AES_SHA_US	SSL 3.0	SHA-1	AES	128	No	No	8.0.0.2
All	DES_SHA_EXPORT[3] [8]	SSL 3.0	SHA-1	DES	56	No	No	8.0.0.2
Linux Windows UNIX	DES_SHA_EXPORT1024[4]	SSL 3.0	SHA-1	DES	56	No	No	8.0.0.2
Linux Windows UNIX	FIPS_WITH_DES_CBC_SHA	SSL 3.0	SHA-1	DES	56	No[6]	No	8.0.0.2
Linux Windows UNIX	FIPS_WITH_3DES_EDE_CBC_SHA	SSL 3.0	SHA-1	3DES	168	No[7]	No	8.0.0.6
All	NULL_MD5	SSL 3.0	MD5	None	0	No	No	8.0.0.2
All	NULL_SHA	SSL 3.0	SHA-1	None	0	No	No	8.0.0.2
All	RC2_MD5_EXPORT[3] [8]	SSL 3.0	MD5	RC2	40	No	No	8.0.0.3
All	RC4_MD5_EXPORT[3]	SSL 3.0	MD5	RC4	40	No	No	8.0.0.3
All	RC4_MD5_US	SSL 3.0	MD5	RC4	128	No	No	8.0.0.3
All	RC4_SHA_US [8]	SSL 3.0	SHA-1	RC4	128	No	No	8.0.0.3
Linux Windows UNIX	RC4_56_SHA_EXPORT1024[4]	SSL 3.0	SHA-1	RC4	56	No	No	8.0.0.3
All	TRIPLE_DES_SHA_US [8]	SSL 3.0	SHA-1	3DES	168	No	No	8.0.0.6
IBM i	TLS_RSA_EXPORT_WITH_RC2_40_MD5	TLS 1.0	MD5	RC2	40	No	No	8.0.0.3
IBM i	TLS_RSA_EXPORT_WITH_RC4_40_MD5[3]	TLS 1.0	MD5	RC4	40	No	No	8.0.0.3

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B	Update when deprecated
All	TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	SHA-1	DES	56	No[5]	No	8.0.0.2
IBM i	TLS_RSA_WITH_NULL_MD5	TLS 1.0	MD5	None	0	No	No	8.0.0.2
IBM i	TLS_RSA_WITH_NULL_SHA	TLS 1.0	SHA-1	None	0	No	No	8.0.0.2
IBM i	TLS_RSA_WITH_RC4_128_MD5	TLS 1.0	MD5	RC4	128	No	No	8.0.0.3
Linux Windows UNIX	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	SHA-1	None	0	No	No	8.0.0.2
Linux Windows UNIX	ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	SHA-1	RC4	128	No	No	8.0.0.3
Linux Windows UNIX	ECDHE_RSA_NULL_SHA256	TLS 1.2	SHA-1	None	0	No	No	8.0.0.2
Linux Windows UNIX	ECDHE_RSA_RC4_128_SHA256	TLS 1.2	SHA-1	RC4	128	No	No	8.0.0.3
Linux Windows UNIX	TLS_RSA_WITH_NULL_NULL	TLS 1.2	None	None	0	No	No	8.0.0.2
All	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	SHA-256	None	0	No	No	8.0.0.3
Linux Windows UNIX	TLS_RSA_WITH_RC4_128_SHA256	TLS 1.2	SHA-1	RC4	128	No	No	8.0.0.3
All	TLS_RSA_WITH_3DES_EDE_CBC_SHA[9]	TLS 1.0	SHA-1	3DES	168	Yes	No	8.0.0.6
Linux Windows UNIX	ECDHE_ECDSA_3DES_EDE_CBC_SHA256[9]	TLS 1.2	SHA-1	3DES	168	Yes	No	8.0.0.6
Linux Windows UNIX	ECDHE_RSA_3DES_EDE_CBC_SHA256[9]	TLS 1.2	SHA-1	3DES	168	Yes	No	8.0.0.6

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B	Update when deprecated

Notes:

1. If no specific platform is noted, the CipherSpec is available on all platforms.
2. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
3. The maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
4. The handshake key size is 1024 bits.
5. This CipherSpec was FIPS 140-2 certified before 19 May 2007.
6. This CipherSpec was FIPS 140-2 certified before 19 May 2007. The name `FIPS_WITH_DES_CBC_SHA` is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. This CipherSpec is deprecated and its use is not recommended.
7. The name `FIPS_WITH_3DES_EDE_CBC_SHA` is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. The use of this CipherSpec is deprecated.
8. These CipherSpecs are no longer supported by IBM MQ classes for Java or IBM MQ classes for JMS. For more information, see SSL/TLS CipherSpecs and CipherSuites in IBM MQ classes for Java or SSL/TLS CipherSpecs and CipherSuites in IBM MQ classes for JMS.
9. This CipherSpec can be used to transfer up to 32 GB of data before the connection is terminated with error AMQ9288. To avoid this error, either avoid using triple DES, or enable secret key reset when using this CipherSpec.

Related concepts:

"Digital certificates and CipherSpec compatibility in IBM MQ" on page 419
This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

Related information:

DEFINE CHANNEL

ALTER CHANNEL

Obtaining information about CipherSpecs using IBM MQ Explorer

You can use IBM MQ Explorer to display descriptions of CipherSpecs.

Use the following procedure to obtain information about the CipherSpecs in "Enabling CipherSpecs" on page 797:

1. Open **IBM MQ Explorer** and expand the **Queue Managers** folder.
2. Ensure that you have started your queue manager.
3. Select the queue manager you want to work with and click **Channels**.
4. Right-click the channel you want to work with and select **Properties**.
5. Select the **SSL** property page.
6. Select from the list the CipherSpec you want to work with. A description is displayed in the window below the list.

Alternatives for specifying CipherSpecs

For those platforms where the operating system provides the SSL support, your system might support new CipherSpecs. You can specify a new CipherSpec with the SSLCIPH parameter, but the value you supply depends on your platform.

Note: This section does not apply to UNIX, Linux or Windows systems, because the CipherSpecs are provided with the IBM MQ product, so new CipherSpecs do not become available after shipment.

For those platforms where the operating system provides the SSL support, your system might support new CipherSpecs that are not included in "Enabling CipherSpecs" on page 797. You can specify a new CipherSpec with the SSLCIPH parameter, but the value you supply depends on your platform. In all cases the specification *must* correspond to an SSL CipherSpec that is both valid and supported by the version of SSL your system is running.

IBM i

A two-character string representing a hexadecimal value.

For more information about the permitted values, refer to the appropriate product documentation (search on *cipher_spec*):

- For V5R3, the iSeries product documentation
- For V5R4, the IBM i product documentation

You can use either the CHGMQMCHL or the CRTMQMCHL command to specify the value, for example:

```
CRTMQMCHL CHLNAME(' channel name ') SSLCIPH(' hexadecimal value ')
```

You can also use the ALTER QMGR MQSC command to set the `SSLCIPH` parameter.

z/OS

A four-character string representing a hexadecimal value. The hexadecimal codes correspond to the values defined in the SSL protocol.

For more information, refer to the Cipher suite definitions in the z/OS Cryptographic Services System SSL Programming, where there is a list of all the supported SSL V3.0, TLS V1.0 and TLS V1.2 cipher specifications in the form of 4-digit hexadecimal codes.

Considerations for IBM MQ clusters

With IBM MQ clusters it is safest to use the CipherSpec names in "Enabling CipherSpecs" on page 797. If you use an alternative specification, be aware that the specification might not be valid on other platforms. For more information, refer to "SSL and clusters" on page 817.

Specifying a CipherSpec for an IBM MQ MQI client

You have three options for specifying a CipherSpec for an IBM MQ MQI client.

These options are as follows:

- Using a channel definition table
- Using the SSL configuration options structure, MQSCO, on an MQCONNX call
- Using the Active Directory (on Windows systems with Active Directory support)

Specifying a CipherSuite with IBM MQ classes for Java and IBM MQ classes for JMS

IBM MQ classes for Java and IBM MQ classes for JMS specify CipherSuites differently from other platforms.

For information about specifying a CipherSuite with IBM MQ classes for Java, see Secure Sockets Layer (SSL) support for Java

For information about specifying a CipherSuite with IBM MQ classes for JMS, see Using Secure Sockets Layer (SSL) with IBM MQ classes for JMS

Specifying a CipherSpec for IBM MQ.NET

For IBM MQ.NET you can specify the CipherSpec either by using the MQEnvironment class or by using the MQC.SSL_CIPHER_SPEC_PROPERTY in the hash table of connection properties.

For information about specifying a CipherSpec for the .NET unmanaged client, see Enabling SSL for the unmanaged .NET client

For information about specifying a CipherSpec for the .NET managed client, see CipherSpec support for the managed .NET client

Resetting SSL and TLS secret keys

IBM MQ supports the resetting of secret keys on queue managers and clients.

Secret keys are reset when a specified number of encrypted bytes of data have flowed across the channel, or after the channel has been idle for a period.

The key reset value is always set by the initiating side of the IBM MQ channel.

Queue manager

For a queue manager, use the command **ALTER QMGR** with the parameter **SSLRKEYC** to set the values used during key renegotiation.

IBM i On IBM i, use **CHGMQM** with the **SSLRSTCNT** parameter.

MQI client

By default, MQI clients do not renegotiate the secret key. You can make an MQI client renegotiate the key in any of three ways. In the following list, the methods are shown in order of priority. If you specify multiple values, the highest priority value is used.

1. By using the KeyResetCount field in the MQSCO structure on an MQCONNX call
2. By using the environment variable MQSSLRESET
3. By setting the SSLKeyResetCount attribute in the MQI client configuration file

These variables can be set to an integer in the range 0 through 999 999 999, representing the number of unencrypted bytes sent and received within an SSL or TLS conversation before the SSL or TLS secret key is renegotiated. Specifying a value of 0 indicates that SSL or TLS secret keys are never renegotiated. If you specify an SSL or TLS secret key reset count in the range 1 byte through 32 KB, SSL or TLS channels will use a secret key reset count of 32 KB. This is to avoid excessive key resets which would occur for small SSL or TLS secret key reset values.

If a value greater than zero is specified and channel heartbeats are enabled for the channel, the secret key is also renegotiated before message data is sent or received following a channel heartbeat.

The count of bytes until the next secret key renegotiation is reset after each successful renegotiation.

For full details of the MQSCO structure, see KeyResetCount (MQLONG). For full details of MQSSLRESET, see MQSSLRESET. For more information about the use of SSL or TLS in the client configuration file, see SSL stanza of the client configuration file.

Java

For IBM MQ classes for Java, an application can reset the secret key in either of the following ways:
- By setting the sslResetCount field in the MQEnvironment class.
- By setting the environment property MQC.SSL_RESET_COUNT_PROPERTY in a Hashtable object. The application then assigns the hashtable to the `properties` field in the MQEnvironment class, or passes the hashtable to an MQQueueManager object on its constructor.

If the application uses more than one of these ways, the usual precedence rules apply. See Class com.ibm.mq.MQEnvironment for the precedence rules.

The value of the sslResetCount field or environment property MQC.SSL_RESET_COUNT_PROPERTY represents the total number of bytes sent and received by the IBM MQ classes for Java client code before the secret key is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by the IBM MQ classes for Java client.

If the reset count is zero, which is the default value, the secret key is never renegotiated. The reset count is ignored if no CipherSuite is specified.

JMS

For IBM MQ classes for JMS, the SSLRESETCOUNT property represents the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by IBM MQ classes for JMS. For example, to configure a ConnectionFactory object that can be used to create a connection over an SSL or TLS enabled MQI channel with a secret key that is renegotiated after 4 MB of data have flowed, issue the following command to JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

If the value of SSLRESETCOUNT is zero, which is the default value, the secret key is never renegotiated. The SSLRESETCOUNT property is ignored if SSLCIPHERSUITE is not set.

.NET

For .NET unmanaged clients, the integer property SSLKeyResetCount indicates the number of unencrypted bytes sent and received within an SSL or TLS conversation before the secret key is renegotiated.

For information about the use of object properties in IBM MQ classes for .NET, see Getting and setting attribute values.

For .NET managed clients, the SSLStream class does not support secret key reset/renegotiation. However, to be consistent with other IBM MQ clients, the IBM MQ managed .NET client allows applications to set SSLKeyResetCount. For more information, see Secret key reset or renegotiation.

XMS .NET

For XMS .NET unmanaged clients, see Secure connections to an IBM WebSphere MQ queue manager.

Related information:

ALTER QMGR

DISPLAY QMGR

Change Message Queue Manager (CHGMQM)

Display Message Queue Manager (DSPMQM)

Implementing confidentiality in user exit programs

Implementing confidentiality in security exits

Security exits can play a role in the confidentiality service by generating and distributing the symmetric key for encrypting and decrypting the data that flows on the channel. A common technique for doing this uses PKI technology.

One security exit generates a random data value, encrypts it with the public key of the queue manager or user that the partner security exit is representing, and sends the encrypted data to its partner in a security message. The partner security exit decrypts the random data value with the private key of the queue manager or user it is representing. Each security exit can now use the random data value to derive the symmetric key independently of the other by using an algorithm known to both of them. Alternatively, they can use the random data value as the key.

If the first security exit has not authenticated its partner by this time, the next security message sent by the partner can contain an expected value encrypted with the symmetric key. The first security exit can now authenticate its partner by checking that the partner security exit was able to encrypt the expected value correctly.

The security exits can also use this opportunity to agree the algorithm for encrypting and decrypting the data that flows on the channel, if more than one algorithm is available for use.

Implementing confidentiality in message exits

A message exit at the sending end of a channel can encrypt the application data in a message and another message exit at the receiving end of the channel can decrypt the data. For performance reasons, a symmetric key algorithm is normally used for this purpose. For more information about how the symmetric key can be generated and distributed, see "Implementing confidentiality in user exit programs."

Headers in a message, such as the transmission queue header, MQXQH, which includes the embedded message descriptor, must not be encrypted by a message exit. This is because data conversion of the message headers takes place either after a message exit is called at the sending end or before a message exit is called at the receiving end. If the headers are encrypted, data conversion fails and the channel stops.

Implementing confidentiality in send and receive exits

Send and receive exits can be used to encrypt and decrypt the data that flows on a channel. They are more appropriate than message exits for providing this service for the following reasons:

- On a message channel, message headers can be encrypted as well as the application data in the messages.

- Send and receive exits can be used on MQI channels as well as message channels. Parameters on MQI calls might contain sensitive application data that needs to be protected while it flows on an MQI channel. You can therefore use the same send and receive exits on both kinds of channels.

Implementing confidentiality in the API exit and API-crossing exit

The application data in a message can be encrypted by an API or API-crossing exit when the message is put by the sending application and decrypted by a second exit when the message is retrieved by the receiving application. For performance reasons, a symmetric key algorithm is typically used for this purpose. However, at the application level, where many users might be sending messages to each other, the problem is how to ensure that only the intended receiver of a message is able to decrypt the message. One solution is to use a different symmetric key for each pair of users that send messages to each other. But this solution might be difficult and time consuming to administer, particularly if the users belong to different organizations. A standard way of solving this problem is known as *digital enveloping* and uses PKI technology.

When an application puts a message on a queue, an API or API-crossing exit generates a random symmetric key and uses the key to encrypt the application data in the message. The exit encrypts the symmetric key with the public key of the intended receiver. It then replaces the application data in the message with the encrypted application data and the encrypted symmetric key. In this way, only the intended receiver can decrypt the symmetric key and therefore the application data. If an encrypted message has more than one possible intended receiver, the exit can encrypt a copy of the symmetric key for each intended receiver.

If different algorithms for encrypting and decrypting the application data are available for use, the exit can include the name of the algorithm it has used.

Data integrity of messages

To maintain data integrity, you can use various types of user exit program to provide message digests or digital signatures for your messages.

Data integrity

Implementing data integrity in messages
When you use SSL or TLS, your choice of CipherSpec determines the level of data integrity in the enterprise. If you use the IBM MQ Advanced Message Service (AMS) you can specify the integrity for a unique message.

Implementing data integrity in message exits

A message can be digitally signed by a message exit at the sending end of a channel. The digital signature can then be checked by a message exit at the receiving end of a channel to detect whether the message has been deliberately modified.

Some protection can be provided by using a message digest instead of a digital signature. A message digest might be effective against casual or indiscriminate tampering, but it does not prevent the more informed individual from changing or replacing the message, and generating a completely new digest for it. This is particularly true if the algorithm that is used to generate the message digest is a well known one.

Implementing data integrity in send and receive exits
On a message channel, message exits are more appropriate for providing this service because a message exit has access to a whole message. On an MQI channel, parameters on MQI calls might contain application data that needs to be protected and only send and receive exits can provide this protection.

Implementing data integrity in the API exit or API-crossing exit

A message can be digitally signed by an API or API-crossing exit when the message is put by the sending application. The digital signature can then be checked by a second exit when the message is retrieved by the receiving application to detect whether the message has been deliberately modified.

Some protection can be provided by using a message digest instead of a digital signature. A message digest might be effective against casual or indiscriminate tampering, but it does not prevent the more informed individual from changing or replacing the message, and generating a completely new digest for it. This is particularly true if the algorithm that is used to generate the message digest is a well known one,

Further information

See the section on “Enabling CipherSpecs” on page 797 for more information on ensuring data integrity.

Related information:
Connecting two queue managers using SSL or TLS
Connecting a client to a queue manager securely

Auditing

You can check for security intrusions, or attempted intrusions, by using event messages. You can also check the security of your system by using the MQ Explorer.

To detect attempts to perform unauthorized actions such as connecting to a queue manager or put a message on a queue, inspect the event messages produced by your queue managers, particularly authority event messages. For more information about queue manager event messages, see Queue manager events, and for more information about event monitoring in general, see Event monitoring.

Keeping clusters secure

Authorize or prevent queue managers joining clusters or putting messages on cluster queues. Force a queue manager to leave a cluster. Take account of some additional considerations when configuring SSL for clusters.

Stopping unauthorized queue managers sending messages

Prevent unauthorized queue managers sending messages to your queue manager using a channel security exit.

Before you begin

Clustering has no effect on the way security exits work. You can restrict access to a queue manager in the same way as you would in a distributed queuing environment.

About this task

Prevent selected queue managers from sending messages to your queue manager:

Procedure

1. Define a channel security exit program on the `CLUSRCVR` channel definition.
2. Write a program that authenticates queue managers trying to send messages on your cluster-receiver channel and denies them access if they are not authorized.

What to do next

Channel security exit programs are called at MCA initiation and termination.

Stopping unauthorized queue managers putting messages on your queues

Use the channel put authority attribute on the cluster-receiver channel to stop unauthorized queue managers putting messages on your queues. Authorize a remote queue manager by checking the user ID in the message using RACF on z/OS, or the OAM on other platforms.

About this task

Use the security facilities of a platform and the access control mechanism in IBM MQ to control access to queues.

Procedure

1. To prevent certain queue managers from putting messages on a queue, use the security facilities available on your platform.

   For example:

   - RACF or other external security managers on IBM MQ for z/OS
   - The object authority manager (OAM) on other platforms.
2. Use the put authority, `PUTAUT`, attribute on the `CLUSRCVR` channel definition.

   The `PUTAUT` attribute allows you to specify what user identifiers are to be used to establish authority to put a message to a queue.

The options on the `PUTAUT` attribute are:

`DEF`
: Use the default user ID. On z/OS, the check might involve using both the user ID received from the network and that derived from `MCAUSER`.

`CTX`
: Use the user ID in the context information associated with the message. On z/OS the check might involve using either the user ID received from the network, or that derived from `MCAUSER`, or both. Use this option if the link is trusted and authenticated.

`ONLYMCA` (z/OS only)
: As for DEF, but any user ID received from the network is not used. Use this option if the link is not trusted. You want to allow only a specific set of actions on it, which are defined for the `MCAUSER`.

`ALTMCA` (z/OS only)
: As for `CTX`, but any user ID received from the network is not used.

Authorizing putting messages on remote cluster queues

On z/OS set up authorization to put to a cluster queue using RACF. On other platforms, authorize access to connect to the queue managers, and to put to the queues on those queue managers.

About this task

The default behavior is to perform access control against the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. Note that this behavior applies, even if you are using multiple transmission queues.

The specific behavior described in this topic applies only when you have configured the **`ClusterQueueAccessControl`** attribute in the `qm.ini` file to be *RQMName*, as described in the Security stanza topic, and restarted the queue manager.

Procedure

- For z/OS, issue the following commands:

```
RDEFINE MQQUEUE QMgrName.QUEUE. QueueName UACC(NONE)
PERMIT QMgrName.QUEUE. QueueName CLASS(MQADMIN) ID(GroupName) ACCESS(UPDATE)
```

- For UNIX, Linux and Windows systems, issue the following commands:

```
setmqaut -m QMgrName -t qmgr -g GroupName +connect
setmqaut -m QMgrName -t queue -n QueueName -g GroupName -all +put
```

- For IBM i, issue the following commands:

```
GRTMQMAUT OBJ(' QMgrName ') OBJTYPE(*MQM) USER(GroupName) AUT(*CONNECT)
GRTMQMAUT OBJ(' QueueName ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
```

The user can put messages only to the specified cluster queue, and no other cluster queues.

The variable names have the following meanings:

QMgrName
: The name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.

GroupName
: The name of the group to be granted access.

QueueName
: Name of the queue or generic profile for which to change authorizations.

What to do next

If you specify a reply-to queue when you put a message on a cluster queue, the consuming application must have authority to send the reply. Set this authority by following the instructions in "Granting authority to put messages to a remote cluster queue" on page 773.

Related information:

Security stanza in qm.ini

Preventing queue managers joining a cluster

If a rogue queue manager joins a cluster it is difficult to prevent it receiving messages you do not want it to receive.

Procedure

If you want to ensure that only certain authorized queue managers join a cluster you have a choice of three techniques:

- Using channel authentication records you can block the cluster channel connection based on: the remote IP address, the remote queue manager name, or the SSL/TLS Distinguished Name provided by the remote system.
- Write an exit program to prevent unauthorized queue managers from writing to `SYSTEM.CLUSTER.COMMAND.QUEUE`. Do not restrict access to `SYSTEM.CLUSTER.COMMAND.QUEUE` such that no queue manager can write to it, or you would prevent any queue manager from joining the cluster.
- A security exit program on the `CLUSRCVR` channel definition.

Security exits on cluster channels

Extra considerations when using security exits on cluster channels.

About this task

When a cluster-sender channel is first started, it uses attributes defined manually by a system administrator. When the channel is stopped and restarted, it picks up the attributes from the corresponding cluster-receiver channel definition. The original cluster-sender channel definition is overwritten with the new attributes, including the `SecurityExit` attribute.

Procedure

1. You must define a security exit on both the cluster-sender end and the cluster-receiver end of a channel.

   The initial connection must be made with a security-exit handshake, even though the security exit name is sent over from the cluster-receiver definition.
2. Validate the `PartnerName` in the `MQCXP` structure in the security exit.

   The exit must allow the channel to start only if the partner queue manager is authorized
3. Design the security exit on the cluster-receiver definition to be receiver initiated.
4. If you design it as sender initiated, an unauthorized queue manager without a security exit can join the cluster because no security checks are performed.

   Not until the channel is stopped and restarted can the `SCYEXIT` name be sent over from the cluster-receiver definition and full security checks made.
5. To view the cluster-sender channel definition that is currently in use, use the command:

   ```
   DISPLAY CLUSQMGR( queue manager ) ALL
   ```

   The command displays the attributes that have been sent across from the cluster-receiver definition.
6. To view the original definition, use the command:

```
DISPLAY CHANNEL( channel name ) ALL
```

7. You might need to define a channel auto-definition exit, `CHADEXIT`, on the cluster-sender queue manager, if the queue managers are on different platforms.

   Use the channel auto-definition exit to set the `SecurityExit` attribute to an appropriate format for the target platform.
8. Deploy and configure the security-exit.

   z/OS **z/OS**

   The security-exit load module must be in the data set specified in the `CSQXLIB DD` statement of the channel-initiator address-space procedure.

   Windows UNIX Linux **Windows, UNIX and Linux systems**

   - The security-exit dynamic link library must be in the path specified in the `SCYEXIT` attribute of the channel definition.
   - The channel auto-definition exit dynamic link library must be in the path specified in the `CHADEXIT` attribute of the queue manager definition.

Forcing unwanted queue managers to leave a cluster

Force an unwanted queue manager to leave a cluster by issuing the `RESET CLUSTER` command at a full repository queue manager.

About this task

You can force an unwanted queue manager to leave a cluster. If for example, a queue manager is deleted but its cluster-receiver channels are still defined to the cluster. You might want to tidy up.

Only full repository queue managers are authorized to eject a queue manager from a cluster.

Note: Although using the RESET CLUSTER command forcibly removes a queue manager from a cluster, the use of RESET CLUSTER by itself does not prevent the queue manager rejoining the cluster later. To ensure that the queue manager does not rejoin the cluster, follow the steps detailed in "Preventing queue managers joining a cluster" on page 815.

Follow this procedure to eject the queue manager `OSLO` from the cluster `NORWAY`:

Procedure

1. On a full repository queue manager, issue the command:
   ```
   RESET CLUSTER(NORWAY) QMNAME(OSLO) ACTION(FORCEREMOVE)
   ```
2. Alternative use the `QMID` instead of `QMNAME` in the command:
   ```
   RESET CLUSTER(NORWAY) QMID(qmid) ACTION(FORCEREMOVE)
   ```

Results

The queue manager that is force removed does not change; its local cluster definitions show it to be in the cluster. The definitions at all other queue managers do not show it in the cluster.

Preventing queue managers receiving messages

You can prevent a cluster queue manager from receiving messages it is unauthorized to receive by using exit programs.

About this task

It is difficult to stop a queue manager that is a member of a cluster from defining a queue. There is a danger that a rogue queue manager joins a cluster, and defines its own instance of one of the queues in the cluster. It can now receive messages that it is not authorized to receive. To prevent a queue manager receiving messages, use one of the following options given in the procedure.

Procedure

- A channel exit program on each cluster-sender channel. The exit program uses the connection name to determine the suitability of the destination queue manager to be sent the messages.
- A cluster workload exit program, which uses the destination records to determine the suitability of the destination queue and queue manager to be sent the messages.

SSL and clusters

When configuring SSL for clusters, be aware a CLUSRCVR channel definition is propagated to other queue managers as an auto-defined CLUSSDR channel. If a CLUSRCVR channel uses SSL, you must configure SSL on all queue managers that communicate using the channel.

For more information about SSL and TLS, see "SSL and TLS security protocols in IBM MQ" on page 397. The advice there is generally applicable to cluster channels, but you might want to give some special consideration to the following:

In an IBM MQ cluster a particular `CLUSRCVR` channel definition is frequently propagated to many other queue managers where it is transformed into an auto-defined `CLUSSDR`. Subsequently the auto-defined `CLUSSDR` is used to start a channel to the `CLUSRCVR`. If the `CLUSRCVR` is configured for SSL connectivity the following considerations apply:

- All queue managers that want to communicate with this `CLUSRCVR` must have access to SSL support. This SSL provision must support the CipherSpec for the channel.
- The different queue managers to which the auto-defined cluster-sender channels have been propagated will each have a different distinguished name associated. If distinguished name peer checking is to be used on the `CLUSRCVR` it must be set up so all of the distinguished names that can be received are successfully matched.

  For example, let us assume that all of the queue managers that will host cluster-sender channels which will connect to a particular `CLUSRCVR`, have certificates associated. Let us also assume that the distinguished names in all of these certificates define the country as UK, organization as IBM, the organization unit as IBM MQ Development, and all have common names in the form `DEVT.QMnnn`, where `nnn` is numeric.

  In this case an `SSLPEER` value of `C=UK, O=IBM, OU=IBM MQ Development, CN=DEVT.QM*` on the `CLUSRCVR` will allow all the required cluster-sender channels to connect successfully, but will prevent unwanted cluster-sender channels from connecting.
- If custom CipherSpec strings are used, be aware that the custom string formats are not allowed on all platforms. An example of this is that the `CipherSpec` string `RC4_SHA_US` has a value of `05` on IBM i but is not a valid specification on UNIX, Linux or Windows systems. So if custom `SSLCIPH` parameters are used on a `CLUSRCVR`, all resulting auto-defined cluster-sender channels should reside on platforms on which the underlying SSL support implements this `CipherSpec` and on which it can be specified with the custom value. If you cannot select a value for the `SSLCIPH` parameter that will be understood

throughout your cluster you will need a channel auto definition exit to change it into something the platforms being used will understand. Use the textual `CipherSpec` strings where possible (for example `RC4_MD5_US`).

An `SSLCRLNL` parameter applies to an individual queue manager and is not propagated to other queue managers within a cluster.

Upgrading clustered queue managers and channels to SSL

Upgrade the cluster channels one at a time, changing all the `CLUSRCVR` channels before the `CLUSSDR` channels.

Before you begin

Consider the following considerations, as these might affect your choice of CipherSpec for a cluster:

- Some CipherSpecs are not available on all platforms. Take care to choose a CipherSpec that is supported by all of the queue managers in the cluster.
- Some CipherSpecs might be new in the current IBM MQ release and not supported in older releases. A cluster containing queue managers running at different MQ releases is only be able to use the CipherSpecs supported by each release.

  To use a new CipherSpec within a cluster, you must first migrate all of the cluster queue managers to the current release.
- Some CipherSpecs require a specific type of digital certificate to be used, notably those that use Elliptic Curve Cryptography.

Upgrade all queue managers in the cluster to IBM MQ V6 or higher, if they are not already at these levels. Distribute the certificates and keys so that SSL works from each of them.

About this task

Change the `CLUSRCVR` channels before the `CLUSSDR` channels.

Procedure

1. Switch the `CLUSRCVR` channels to SSL in any order you like, changing one `CLUSRCVR` at a time, and allow the changes to flow through the cluster before changing the next.

   Important: Make sure that you do not change the reverse path until the changes for the current channel have been distributed throughout the cluster.
2. Optional: Switch all manual `CLUSSDR` channels to SSL. This does not have any effect on the operation of the cluster, unless you use the `REFRESH CLUSTER` command with the `REPOS(YES)` option.

   Note: For large clusters, use of the **`REFRESH CLUSTER`** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.
3. Use the DISPLAY CLUSQMGR command to ensure that the new security configuration has been propagated throughout the cluster.
4. Restart the channels to use SSL or TLS and run REFRESH SECURITY (SSL).

Related concepts:

"Enabling CipherSpecs" on page 797
Enable a CipherSpec by using the **`SSLCIPH`** parameter in either the **`DEFINE CHANNEL`** MQSC command or the **`ALTER CHANNEL`** MQSC command.

"Digital certificates and CipherSpec compatibility in IBM MQ" on page 419
This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

Related information:

Clustering: Using REFRESH CLUSTER best practices

Disabling TLS on clustered queue managers and channels

To turn off SSL or TLS, set the `SSLCIPH` parameter to `' '`. Disable TLS on the cluster channels individually, changing all the cluster receiver channels before the cluster sender channels.

About this task

Change one cluster receiver channel at a time, and allow the changes to flow through the cluster before changing the next.

Important: Ensure that you do not change the reverse path until the changes for the current channel have been distributed throughout the cluster.

Procedure

1. Set the value of the `SSLCIPH` parameter to `' '`, an empty string in a single quotation mark IBM i , or `*NONE` on IBM i . You can turn off TLS on the cluster receiver channels in any order you like.

   Note that the changes flow in the opposite direction over channels on which you leave TLS active.
2. Check that the new value is reflected in all the other queue managers by using the command **`DISPLAY CLUSQMGR(*)`** `ALL`.
3. Turn off TLS on all manual cluster sender channels. This does not have any effect on the operation of the cluster, unless you use the **`REFRESH CLUSTER`** command with the `REPOS(YES)` option.

   For large clusters, use of the **`REFRESH CLUSTER`** command can be disruptive to the cluster while it is in progress, and again at regular intervals thereafter, when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster for more information.
4. Stop and restart the cluster sender channels.

Publish/subscribe security

The components and interactions that are involved in publish/subscribe are described as an introduction to the more detailed explanations and examples that follow.

There are a number of components involved in publishing and subscribing to a topic. Some of the security relationships between them are illustrated in Figure 75 and described in the following example.

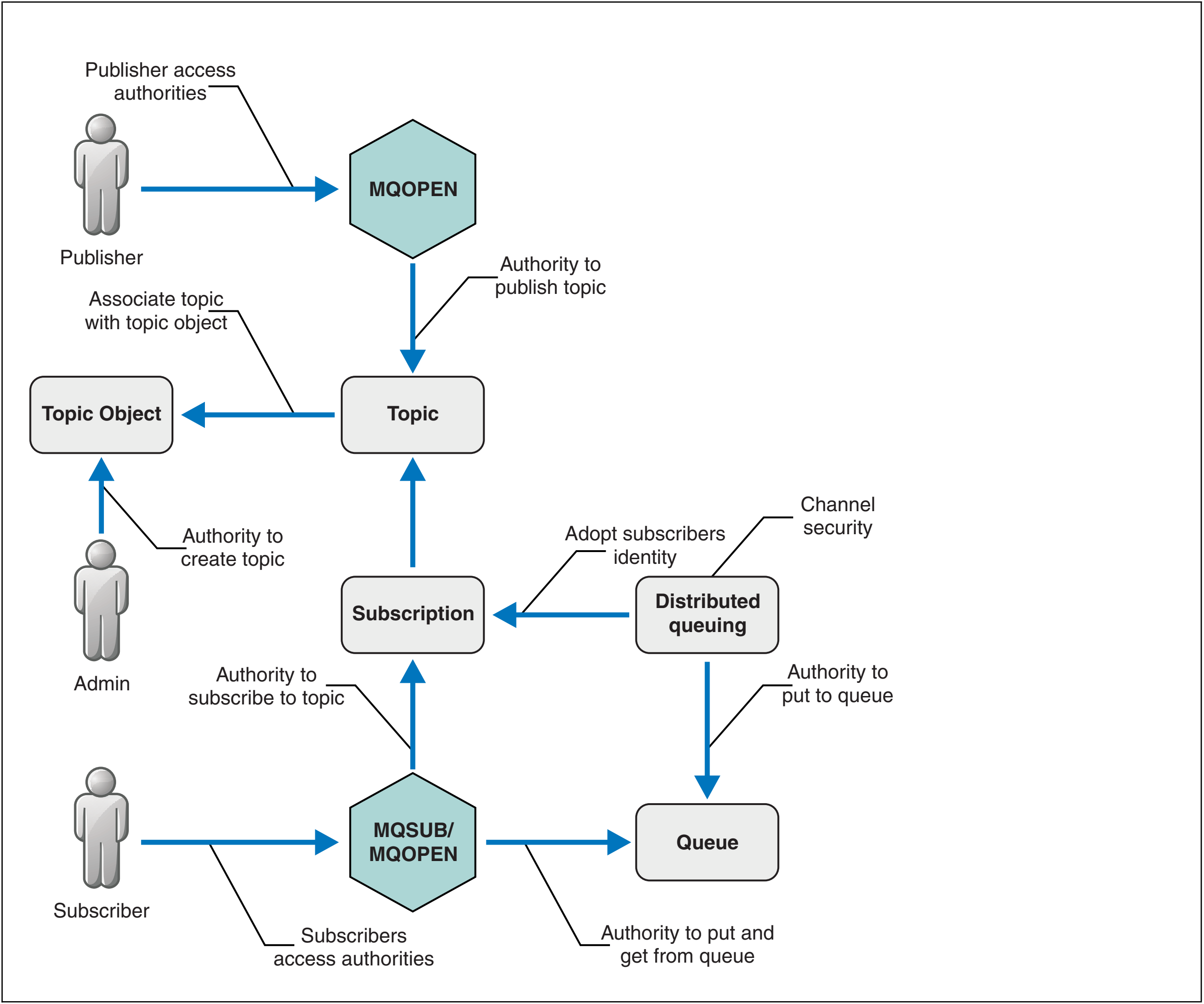

Publisher access authorities
Publisher
MQOPEN
Authority to publish topic
Associate topic with topic object
Topic Object
Topic
Authority to create topic
Admin
Channel security
Adopt subscribers identity
Subscription
Distributed queuing
Authority to subscribe to topic
Authority to put to queue
Subscriber
MQSUB/ MQOPEN
Queue
Subscribers access authorities
Authority to put and get from queue

Figure 75. Publish/subscribe security relationships between various components

Topics
: Topics are identified by topic strings, and are typically organized into trees, see Topic trees. You need to associate a topic with a topic object to control access to the topic. “Topic security model” on page 823 explains how you secure topics using topic objects.

Administrative topic objects
: You can control who has access to a topic, and for what purpose, by using the command **`setmqaut`** with a list of administrative topic objects. See the examples, “Grant access to a user to

subscribe to a topic" on page 828 and "Grant access to a user to publish to a topic" on page 835.

z/OS For controlling access to topic objects on z/OS, see Profiles for topic security.

Subscriptions

Subscribe to one or more topics by creating a subscription supplying a topic string, which can include wildcards, to match against the topic strings of publications. For further details, see:

Subscribe using a topic object
"Subscribing using the topic object name" on page 824

Subscribe using a topic
"Subscribing using a topic string where the topic node does not exist" on page 825

Subscribe using a topic with wildcards
"Subscribing using a topic string that contains wildcard characters" on page 826

A subscription contains information about the identity of the subscriber and the identity of the destination queue on to which the publications are to be placed. It also contains information about how the publication is to be placed on the destination queue.

As well as defining which subscribers have the authority to subscribe to certain topics, you can restrict subscriptions to being used by an individual subscriber. You can also control what information about the subscriber is used by the queue manager when publications are placed on to the destination queue. See "Subscription security" on page 840.

Queues

The destination queue is an important queue to secure. It is local to the subscriber, and publications that matched the subscription are placed onto it. You need to consider access to the destination queue from two perspectives:

1. Putting a publication on to the destination queue.
2. Getting the publication off the destination queue.

The queue manager puts a publication onto the destination queue using an identity provided by the subscriber. The subscriber, or a program that has been delegated the task of getting publications, takes messages off the queue. See "Authority to destination queues" on page 826.

There are no topic object aliases, but you can use an alias queue as the alias for a topic object. If you do so, as well as checking authority to use the topic for publish or subscribe, the queue manager checks authority to use the queue.

"Publish/subscribe security between queue managers" on page 842

Your permission to publish or subscribe to a topic is checked on the local queue manager using local identities and authorizations. Authorization does not depend on whether the topic is defined or not, nor where it is defined. Consequently, you need to perform topic authorization on every queue manager in a cluster when clustered topics are used.

Note: The security model for topics differs from the security model for queues. You can achieve the same result for queues by defining a queue alias locally for every clustered queue.

Queue managers exchange subscriptions in a cluster. In most IBM MQ cluster configurations, channels are configured with *PUTAUT* = *DEF* to place messages onto target queues using the authority of the channel process. You can modify the channel configuration to use *PUTAUT* = *CTX* to require the subscribing user to have authority to propagate a subscription onto another queue manager in a cluster.

"Publish/subscribe security between queue managers" on page 842 describes how to change your channel definitions to control who is allowed to propagate subscriptions onto other servers in the cluster.

Authorization

You can apply authorization to topic objects, just like queues and other objects. There are three authorization operations, `pub`, `sub`, and `resume` that you can apply only to topics. The details are described in Specifying authorities for different object types.

Function calls

In publish and subscribe programs, like in queued programs, authorization checks are made when objects are opened, created, changed, or deleted. Checks are not made when MQPUT or MQGET MQI calls are made to put and get publications.

To publish a topic, perform an MQOPEN on the topic, which performs the authorization checks. Publish messages to the topic handle using the MQPUT command, which performs no authorization checks.

To subscribe to a topic, typically you perform an MQSUB command to create or resume the subscription, and also to open the destination queue to receive publications. Alternatively, perform a separate MQOPEN to open the destination queue, and then perform the MQSUB to create or resume the subscription.

Whichever calls you use, the queue manager checks that you can subscribe to the topic and get the resulting publications from the destination queue. If the destination queue is unmanaged, authorization checks are also made that the queue manager is able to place publications on the destination queue. It uses the identity it adopted from a matching subscription. It is assumed that the queue manager is always able to place publications onto managed destination queues.

Roles

Users are involved in four roles in running publish/subscribe applications:

1. Publisher
2. Subscriber
3. Topic administrator
4. IBM MQ Administrator - member of group `mqm`

Define groups with appropriate authorizations corresponding to the publish, subscribe, and topic administration roles. You can then assign principals to these groups authorizing them to perform specific publish and subscribe tasks.

In addition, you need to extend the administrative operations authorizations to the administrator of the queues and channels responsible for moving publications and subscriptions.

Topic security model

Only defined topic objects can have associated security attributes. For a description of topic objects, see Administrative topic objects. The security attributes specify whether a specified user ID, or security group, is permitted to perform a subscribe or a publish operation on each topic object.

The security attributes are associated with the appropriate administration node in the topic tree. When an authority check is made for a particular user ID during a subscribe or publish operation, the authority granted is based on the security attributes of the associated topic tree node.

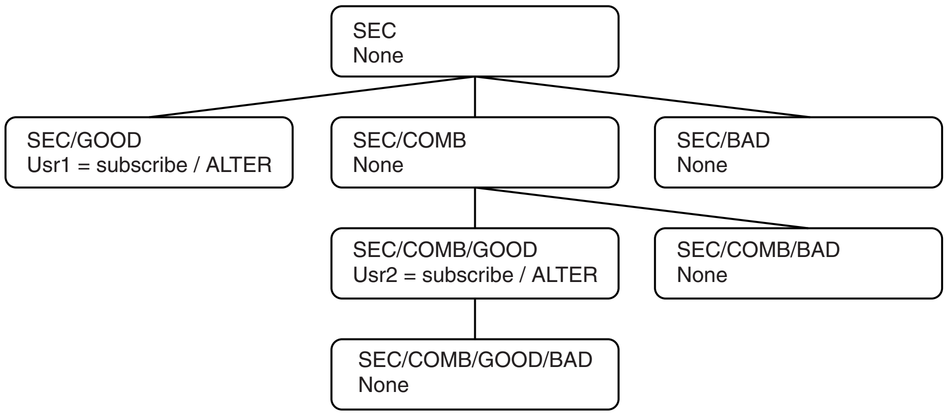
The security attributes are an access control list, indicating what authority a particular operating system user ID or security group has to the topic object.

Consider the following example where the topic objects have been defined with the security attributes, or authorities shown:

Table 84. Example topic object authorities

Topic name	Topic string	Authorities - not z/OS	z/OS authorities
SECROOT	SEC	None	None
SECGOOD	SEC/GOOD	usr1+subscribe	ALTER HLQ.SUBSCRIBE.SECGOOD
SECBAD	SEC/BAD	None	None HLQ.SUBSCRIBE.SECBAD
SECCOMB	SEC/COMB	None	None HLQ.SUBSCRIBE.SECCOMB
SECCOMBB	SEC/COMB/GOOD/BAD	None	None HLQ.SUBSCRIBE.SECCOMBB
SECCOMBG	SEC/COMB/GOOD	usr2+subscribe	ALTER HLQ.SUBSCRIBE.SECCOMBG
SECCOMBN	SEC/COMB/BAD	None	None HLQ.SUBSCRIBE.SECCOMBN

The topic tree with the associated security attributes at each node can be represented as follows:



The examples listed give the following authorizations:
- At the root node of the tree /SEC, no user has authority at that node.
- usr1 has been granted subscribe authority to the object /SEC/GOOD
- usr2 has been granted subscribe authority to the object /SEC/COMB/GOOD

Subscribing using the topic object name

When subscribing to a topic object by specifying the MQCHAR48 name, the corresponding node in the topic tree is located. If the security attributes associated with the node indicate that the user has authority to subscribe, then access is granted.

If the user is not granted access, the parent node in the tree determines if the user has authority to subscribe at the parent node level. If so, then access is granted. If not, then the parent of that node is considered. The recursion continues until a node is located that grants subscribe authority to the user. The recursion stops when the root node is considered without authority having been granted. In the latter case, access is denied.

In short, if any node in the path grants authority to subscribe to that user or application, the subscriber is allowed to subscribe at that node, or anywhere below that node in the topic tree.

The root node in the example is `SEC`.

The user is granted subscribe authority if the access control list indicates that the user ID itself has authority, or that an operating system security group of which the user ID is a member has authority.

So, for example:
- If `usr1` tries to subscribe, using a topic string of `SEC/GOOD`, the subscription would be allowed as the user ID has access to the node associated with that topic. However, if `usr1` tried to subscribe using topic string `SEC/COMB/GOOD` the subscription would not be allowed as the user ID does not have access to the node associated with it.
- If `usr2` tries to subscribe, using a topic string of `SEC/COMB/GOOD` the subscription would be allowed to as the user ID has access to the node associated with the topic. However, if `usr2` tried to subscribe to `SEC/GOOD` the subscription would not be allowed as the user ID does not have access to the node associated with it.
- If `usr2` tries to subscribe using a topic string of `SEC/COMB/GOOD/BAD` the subscription would be allowed to because the user ID has access to the parent node `SEC/COMB/GOOD`.
- If `usr1` or `usr2` tries to subscribe using a topic string of `/SEC/COMB/BAD`, neither would be allowed as they do not have access to the topic node associated with it, or the parent nodes of that topic.

A subscribe operation specifying the name of a topic object that does not exist results in an MQRC_UNKNOWN_OBJECT_NAME error.

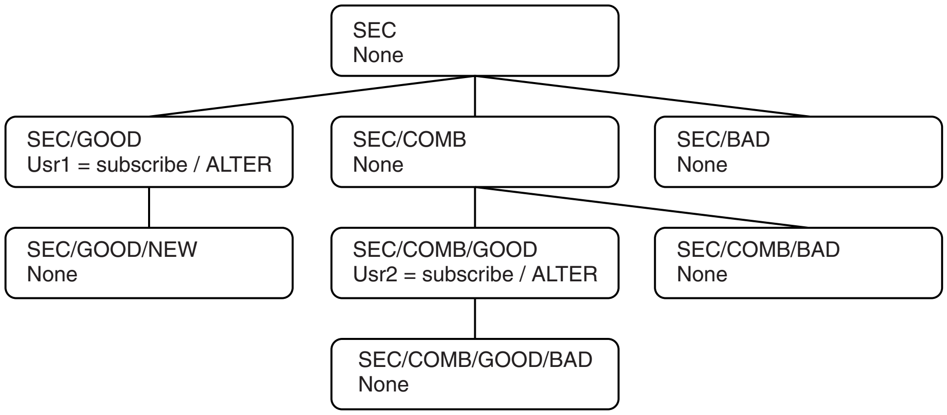
Subscribing using a topic string where the topic node exists

The behavior is the same as when specifying the topic by the MQCHAR48 object name.

Subscribing using a topic string where the topic node does not exist

Consider the case of an application subscribing, specifying a topic string representing a topic node that does not currently exist in the topic tree. The authority check is performed as outlined in the previous section. The check starts with the parent node of that which is represented by the topic string. If the authority is granted, a new node representing the topic string is created in the topic tree.

For example, `usr1` tries to subscribe to a topic `SEC/GOOD/NEW`. Authority is granted as `usr1` has access to the parent node `SEC/GOOD`. A new topic node is created in the tree as the following diagram shows. The new topic node is not a topic object it does not have any security attributes associated with it directly; the attributes are inherited from its parent.



Subscribing using a topic string that contains wildcard characters

Consider the case of subscribing using a topic string that contains a wildcard character. The authority check is made against the node in the topic tree that matches the fully qualified part of the topic string.

So, if an application subscribes to `SEC/COMB/GOOD/*`, an authority check is carried out as outlined in the previous two sections on the node `SEC/COMB/GOOD` in the topic tree.

Similarly, if an application needs to subscribe to `SEC/COMB/*/GOOD`, an authority check is carried out on the node `SEC/COMB`.

Authority to destination queues

When subscribing to a topic, one of the parameters is the handle `hobj` of a queue that has been opened for output to receive the publications.

If `hobj` is not specified, but is blank, a managed queue is created if the following conditions apply:

- The `MQSO_MANAGED` option has been specified.
- The subscription does not exist.
- Create is specified.

If `hobj` is blank, and you are altering or resuming an existing subscription, the previously provided destination queue could be either managed or unmanaged.

The application or user making the MQSUB request must have the authority to put messages to the destination queue it has provided; in effect authority to have published messages put on that queue. The authority check follows the existing rules for queue security checking.

The security checking includes alternate user ID and context security checks where required. To be able to set any of the Identity context fields you must specify the `MQSO_SET_IDENTITY_CONTEXT` option as well as the `MQSO_CREATE` or `MQSO_ALTER` option. You cannot set any of the Identity context fields on an `MQSO_RESUME` request.

If the destination is a managed queue, no security checks are performed against the managed destination. If you are allowed to subscribe to a topic it is assumed that you can use managed destinations.

Publishing using the topic name or topic string where the topic node exists

The security model for publishing is the same as that for subscribing, with the exception of wildcards. Publications do not contain wildcards; so there is no case of a topic string containing wildcards to consider.

The authorities to publish and subscribe are distinct. A user or group can have the authority to do one without necessarily being able to do the other.

When publishing to a topic object by specifying either the MQCHAR48 name or the topic string, the corresponding node in the topic tree is located. If the security attributes associated with the topic node indicates that the user has authority to publish, then access is granted.

If access is not granted, the parent node in the tree determines if the user has authority to publish at that level. If so, then access is granted. If not, the recursion continues until a node is located which grants publish authority to the user. The recursion stops when the root node is considered without authority having been granted. In the latter case, access is denied.

In short, if any node in the path grants authority to publish to that user or application, the publisher is allowed to publish at that node or anywhere below that node in the topic tree.

Publishing using the topic name or topic string where the topic node does not exist

As with the subscribe operation, when an application publishes, specifying a topic string representing a topic node that does not currently exist in the topic tree, the authority check is performed starting with the parent of the node represented by the topic string. If the authority is granted, a new node representing the topic string is created in the topic tree.

Publishing using an alias queue that resolves to a topic object

If you publish using an alias queue that resolves to a topic object then security checking occurs on both the alias queue and the underlying topic to which it resolves.

The security check on the alias queue verifies that the user has authority to put messages on that alias queue and the security check on the topic verifies that the user can publish to that topic. When an alias queue resolves to another queue, checks are *not* made on the underlying queue. Authority checking is performed differently for topics and queues.

Closing a subscription

There is additional security checking if you close a subscription using the `MQCO_REMOVE_SUB` option if you did not create the subscription under this handle.

A security check is performed to ensure that you have the correct authority to do this as the action results in the removal of the subscription. If the security attributes associated with the topic node indicate that the user has authority, then access is granted. If not, then the parent node in the tree is considered to determine if the user has authority to close the subscription. The recursion continues until either authority is granted or the root node is reached.

Defining, altering, and deleting a subscription

No subscribe security checks are performed when a subscription is created administratively, rather than using an MQSUB API request. The administrator has already been given this authority through the command.

Security checks are performed to ensure that publications can be put on the destination queue associated with the subscription. The checks are performed in the same way as for an MQSUB request.

The user ID that is used for these security checks depends upon the command being issued. If the **SUBUSER** parameter is specified it affects the way the check is performed, as shown in Table 85:

Table 85. User IDs used for security checks for commands

Command	SUBUSER specified and blank	SUBUSER specified and completed	SUBUSER not specified
`DEFINE LIKE`	Use the administrator ID	Use the user ID specified in **SUBUSER**	Use the user ID from the LIKE subscription
`DEFINE with no LIKE`	Use the administrator ID	Use the user ID specified in **SUBUSER**	Use the user ID from the SYSTEM.DEFAULT.SUB subscription - if blank, use the administrator ID
`ALTER`	Use the administrator ID	Use the user ID specified in **SUBUSER**	Use the user ID from the existing subscription

The only security check performed when deleting subscriptions using the DELETE SUB command is the command security check.

Example publish/subscribe security setup

This section describes a scenario that has access control setup on topics in a way that allows the security control to be applied as required.

Grant access to a user to subscribe to a topic

This topic is the first one in a list of tasks that tells you how to grant access to topics by more than one user.

About this task

This task assumes that no administrative topic objects exist, nor have any profiles been defined for subscription or publication. The applications are creating new subscriptions, rather than resuming existing ones, and are doing so using the topic string only.

An application can make a subscription by providing a topic object, or a topic string, or a combination of both. Whichever way the application selects, the effect is to make a subscription at a certain point in the topic tree. If this point in the topic tree is represented by an administrative topic object, a security profile is checked based on the name of that topic object.

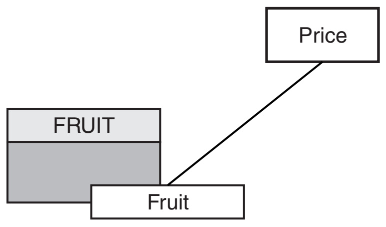


Figure 76. Topic object access example

Table 86. Example topic object access

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(FRUIT) TOPICSTR('Price/Fruit')`.
2. Grant access as follows:
   - z/OS **z/OS** :

     Grant access to `USER1` to subscribe to topic `"Price/Fruit"` by granting the user access to the `hlq.SUBSCRIBE.FRUIT` profile. Do this, using the following RACF commands:
     ```
     RDEFINE MXTOPIC hlq.SUBSCRIBE.FRUIT UACC(NONE)
     PERMIT hlq.SUBSCRIBE.FRUIT CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
     ```
   - Other platforms:

     Grant access to `USER1` to subscribe to topic `"Price/Fruit"` by granting the user access to the `FRUIT` object. Do this, using the authorization command for the platform:

     Windows UNIX Linux **Windows, UNIX and Linux systems**
     ```
     setmqaut -t topic -n FRUIT -p USER1 +sub
     ```

     IBM i **IBM i**
     ```
     GRTMQAUT OBJ(FRUIT) OBJTYPE(*TOPIC) USER(USER1) AUT(*SUB)
     ```

Results

When `USER1` attempts to subscribe to topic `"Price/Fruit"` the result is success.

When `USER2` attempts to subscribe to topic `"Price/Fruit"` the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:
  ```
  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.FRUIT ...

  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
  ```
- Windows UNIX Linux On other platforms, the following authorization event:
  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_SUB_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
  TopicString     "Price/Fruit"
  ```
- IBM i On IBMi, the following authorization event:
  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_SUB_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
  TopicString     "Price/Fruit"
  ```

Note that this is an illustration of what you see; not all the fields.

Grant access to a user to subscribe to a topic deeper within the tree

This topic is the second in a list of tasks that tells you how to grant access to topics by more than one user.

Before you begin

This topic uses the setup described in “Grant access to a user to subscribe to a topic” on page 828.

About this task

If the point in the topic tree where the application makes the subscription is not represented by an administrative topic object, move up the tree until the closest parent administrative topic object is located. The security profile is checked, based on the name of that topic object.

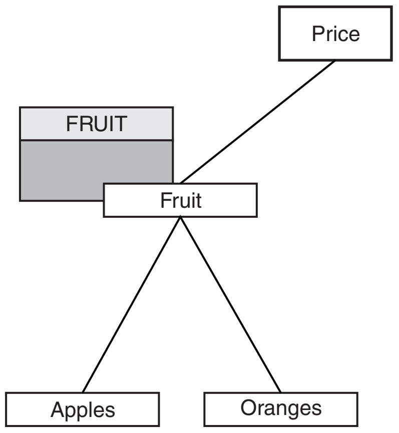


Figure 77. Example of granting access to a topic within a topic tree

Table 87. Access requirements for example topics and topic objects

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT
Price/Fruit/Apples	USER1	
Price/Fruit/Oranges	USER1	

In the previous task `USER1` was granted access to subscribe to topic `"Price/Fruit"` by granting it access to the `hlq.SUBSCRIBE.FRUIT` profile on z/OS and subscribe access to the `FRUIT` profile on other platforms. This single profile also grants `USER1` access to subscribe to `"Price/Fruit/Apples"`, `"Price/Fruit/Oranges"` and `"Price/Fruit/#"`.

When `USER1` attempts to subscribe to topic `"Price/Fruit/Apples"` the result is success.

When `USER2` attempts to subscribe to topic `"Price/Fruit/Apples"` the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
```

- On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier   MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
TopicString     "Price/Fruit/Apples"
```

Note the following:

- The messages you receive on z/OS are identical to those received in the previous task as the same topic objects and profiles are controlling the access.
- The event message you receive on other platforms is similar to the one received in the previous task, but the actual topic string is different.

Grant another user access to subscribe to only the topic deeper within the tree

This topic is the third in a list of tasks that tells you how to grant access to subscribe to topics by more than one user.

Before you begin

This topic uses the setup described in “Grant access to a user to subscribe to a topic deeper within the tree” on page 830.

About this task

In the previous task USER2 was refused access to topic "Price/Fruit/Apples". This topic tells you how to grant access to that topic, but not to any other topics.

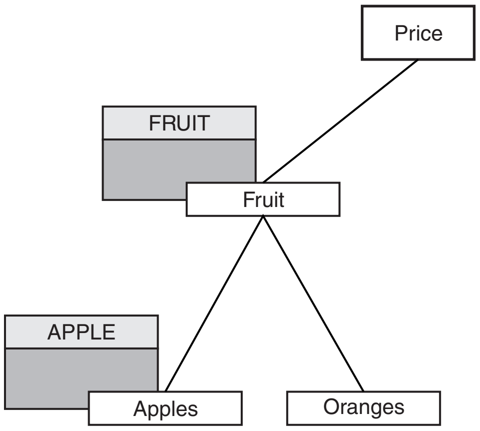


Figure 78. Granting access to specific topics within a topic tree

Table 88. Access requirements for example topics and topic objects

Topic	Subscribe access required	Topic object
Price	No user	None

Table 88. Access requirements for example topics and topic objects (continued)

Topic	Subscribe access required	Topic object
Price/Fruit	USER1	FRUIT
Price/Fruit/Apples	USER1 and USER2	APPLE
Price/Fruit/Oranges	USER1	

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(APPLE) TOPICSTR('Price/Fruit/Apples')`.
2. Grant access as follows:
   - z/OS **z/OS** :

     In the previous task `USER1` was granted access to subscribe to topic `"Price/Fruit/Apples"` by granting the user access to the `hlq.SUBSCRIBE.FRUIT` profile.

     This single profile also granted `USER1` access to subscribe to `"Price/Fruit/Oranges"` `"Price/Fruit/#"` and this access remains even with the addition of the new topic object and the profiles associated with it.

     Grant access to `USER2` to subscribe to topic `"Price/Fruit/Apples"` by granting the user access to the `hlq.SUBSCRIBE.APPLE` profile. Do this, using the following RACF commands:

     ```
     RDEFINE MXTOPIC hlq.SUBSCRIBE.APPLE UACC(NONE)
     PERMIT hlq.SUBSCRIBE.FRUIT APPLE(MXTOPIC) ID(USER2) ACCESS(ALTER)
     ```
   - Other platforms:

     In the previous task `USER1` was granted access to subscribe to topic `"Price/Fruit/Apples"` by granting the user subscribe access to the `FRUIT` profile.

     This single profile also granted `USER1` access to subscribe to `"Price/Fruit/Oranges"` and `"Price/Fruit/#"`, and this access remains even with the addition of the new topic object and the profiles associated with it.

     Grant access to `USER2` to subscribe to topic `"Price/Fruit/Apples"` by granting the user subscribe access to the `APPLE` profile. Do this, using the authorization command for the platform:

     Windows UNIX Linux **Windows, UNIX and Linux systems**

     ```
     setmqaut -t topic -n APPLE -p USER2 +sub
     ```

     IBM i **IBM i**

     ```
     GRTMQAUT OBJ(APPLE) OBJTYPE(*TOPIC) USER(USER2) AUT(*SUB)
     ```

Results

On z/OS, when `USER1` attempts to subscribe to topic `"Price/Fruit/Apples"` the first security check on the `hlq.SUBSCRIBE.APPLE` profile fails, but on moving up the tree the `hlq.SUBSCRIBE.FRUIT` profile allows USER1 to subscribe, so the subscription succeeds and no return code is sent to the MQSUB call. However, a RACF `ICH` message is generated for the first check:

```
ICH408I USER(USER1   ) ...
  hlq.SUBSCRIBE.APPLE ...
```

When `USER2` attempts to subscribe to topic `"Price/Fruit/Apples"` the result is success because the security check passes on the first profile.

When `USER2` attempts to subscribe to topic `"Price/Fruit/Oranges"` the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

  ```
  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.FRUIT ...

  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
  ```

- Windows UNIX Linux On Windows, UNIX and Linux platforms, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_SUB_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
  TopicString     "Price/Fruit/Oranges"
  ```

- IBM i On IBMi, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_SUB_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
  TopicString     "Price/Fruit/Oranges"
  ```

The disadvantage of this setup is that, on z/OS, you receive additional `ICH` messages on the console. You can avoid this if you secure the topic tree in a different manner.

Change access control to avoid additional messages

This topic is the fourth in a list of tasks that tells you how to grant access to subscribe to topics by more than one user and to avoid additional RACF ICH408I messages on z/OS.

Before you begin

This topic enhances the setup described in "Grant another user access to subscribe to only the topic deeper within the tree" on page 831 so that you avoid additional error messages.

About this task

This topic tells you how to grant access to topics deeper in the tree, and how to remove access to the topic lower down the tree when no user requires it.

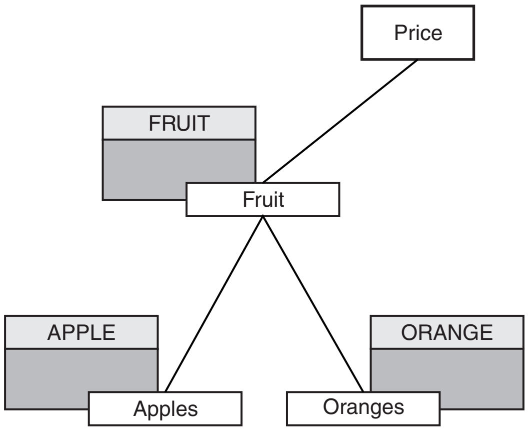


Figure 79. Example of granting access control to avoid additional messages.

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(ORANGE) TOPICSTR('Price/Fruit/Oranges')`.
2. Grant access as follows:
   - z/OS **z/OS** :

     Define a new profile and add access to that profile, and the existing profiles. Do this, using the following RACF commands:

     ```
     RDEFINE MXTOPIC hlq.SUBSCRIBE.ORANGE UACC(NONE)
     PERMIT hlq.SUBSCRIBE.ORANGE CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
     PERMIT hlq.SUBSCRIBE.APPLE CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
     ```
   - Other platforms:

     Setup the equivalent access by using the authorization commands for the platform:

     Windows UNIX Linux **Windows, UNIX and Linux systems**

     ```
     setmqaut -t topic -n ORANGE -p USER1 +sub
     setmqaut -t topic -n APPLE -p USER1 +sub
     ```

     IBM i **IBM i**

     ```
     GRTMQAUT OBJ(ORANGE) OBJTYPE(*TOPIC) USER(USER1) AUT(*SUB)
     GRTMQAUT OBJ(APPLE) OBJTYPE(*TOPIC) USER(USER1) AUT(*SUB)
     ```

Results

On z/OS, when `USER1` attempts to subscribe to topic `"Price/Fruit/Apples"` the first security check on the `hlq.SUBSCRIBE.APPLE` profile succeeds.

Similarly, when `USER2` attempts to subscribe to topic `"Price/Fruit/Apples"` the result is success because the security check passes on the first profile.

When `USER2` attempts to subscribe to topic `"Price/Fruit/Oranges"` the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

  ```
  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.ORANGE ...

  ICH408I USER(USER2   ) ...
  ```

```
  hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
```

- Windows UNIX Linux On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier   MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   ORANGE, FRUIT, SYSTEM.BASE.TOPIC
TopicString     "Price/Fruit/Oranges"
```

- IBM i On IBMi, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier   MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   ORANGE, FRUIT, SYSTEM.BASE.TOPIC
TopicString     "Price/Fruit/Oranges"
```

Grant access to a user to publish to a topic

This topic is the first one in a list of tasks that tells you how to grant access to publish topics by more than one user.

About this task

This task assumes that no administrative topic objects exist on the right hand side of the topic tree, nor have any profiles been defined for publication. The assumption used is that publishers are using the topic string only.

An application can publish to a topic by providing a topic object, or a topic string, or a combination of both. Whichever way the application selects, the effect is to publish at a certain point in the topic tree. If this point in the topic tree is represented by an administrative topic object, a security profile is checked based on the name of that topic object. For example:

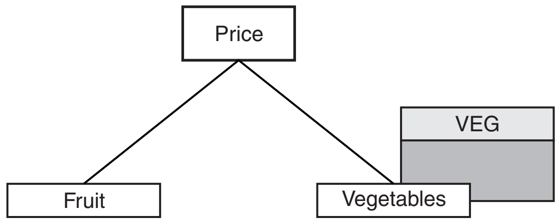


Figure 80. Granting publish access to a topic

Table 89. Example publish access requirements

Topic	Publish access required	Topic object
Price	No user	None
Price/Vegetables	USER1	VEG

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(VEG) TOPICSTR('Price/Vegetables')`.
2. Grant access as follows:

- z/OS **z/OS** :

  Grant access to USER1 to publish to topic "Price/Vegetables" by granting the user access to the hlq.PUBLISH.VEG profile. Do this, using the following RACF commands:

  ```
  RDEFINE MXTOPIC hlq.PUBLISH.VEG UACC(NONE)
  PERMIT hlq.PUBLISH.VEG CLASS(MXTOPIC) ID(USER1) ACCESS(UPDATE)
  ```
- Other platforms:

  Grant access to USER1 to publish to topic "Price/Vegetables" by granting the user access to the VEG profile. Do this, using the authorization command for the platform:

  Windows UNIX Linux **Windows, UNIX and Linux systems**

  ```
  setmqaut -t topic -n VEG -p USER1 +pub
  ```

  IBM i **IBM i**

  ```
  GRTMQAUT OBJ(VEG) OBJTYPE(*TOPIC) USER(USER1) AUT(*PUB)
  ```

Results

When USER1 attempts to publish to topic "Price/Vegetables" the result is success; that is, the MQOPEN call succeeds.

When USER2 attempts to publish to topic "Price/Vegetables" the MQOPEN call fails with an MQRC_NOT_AUTHORIZED message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

  ```
  ICH408I USER(USER2   ) ...
    hlq.PUBLISH.VEG ...

  ICH408I USER(USER2   ) ...
    hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
  ```
- Windows UNIX Linux On other platforms, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_OPEN_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   VEG, SYSTEM.BASE.TOPIC
  TopicString     "Price/Vegetables"
  ```
- IBM i On IBMi, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_OPEN_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   VEG, SYSTEM.BASE.TOPIC
  TopicString     "Price/Vegetables"
  ```

Note that this is an illustration of what you see; not all the fields.

Grant access to a user to publish to a topic deeper within the tree

This topic is the second in a list of tasks that tells you how to grant access to publish to topics by more than one user.

Before you begin

This topic uses the setup described in “Grant access to a user to publish to a topic” on page 835.

About this task

If the point in the topic tree where the application publishes is not represented by an administrative topic object, move up the tree until the closest parent administrative topic object is located. The security profile is checked, based on the name of that topic object.

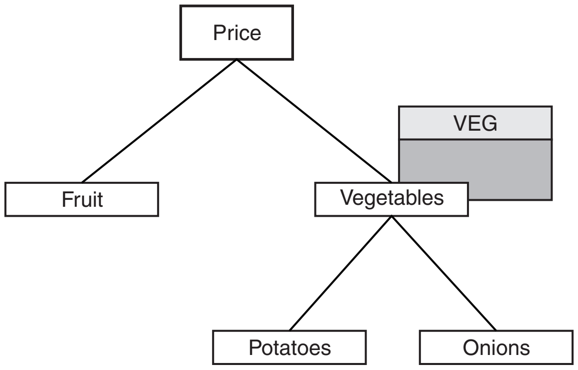


Figure 81. Granting publish access to a topic within a topic tree

Table 90. Example publish access requirements

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Vegetables	USER1	VEG
Price/Vegetables/Potatoes	USER1	
Price/Vegetables/Onions	USER1	

In the previous task `USER1` was granted access to publish topic `"Price/Vegetables/Potatoes"` by granting it access to the `hlq.PUBLISH.VEG` profile on z/OS or publish access to the `VEG` profile on other platforms. This single profile also grants `USER1` access to publish at `"Price/Vegetables/Onions"`.

When `USER1` attempts to publish at topic `"Price/Vegetables/Potatoes"` the result is success; that is the MQOPEN call succeeds.

When `USER2` attempts to subscribe to topic `"Price/Vegetables/Potatoes"` the result is failure; that is, the MQOPEN call fails with an `MQRC_NOT_AUTHORIZED` message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2   ) ...
  hlq.PUBLISH.VEG ...

ICH408I USER(USER2   ) ...
  hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
```

- On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier   MQRQ_OPEN_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   VEG, SYSTEM.BASE.TOPIC
TopicString     "Price/Vegetables/Potatoes"
```

Note the following:

- The messages you receive on z/OS are identical to those received in the previous task as the same topic objects and profiles are controlling the access.
- The event message you receive on other platforms is similar to the one received in the previous task, but the actual topic string is different.

Grant access for publish and subscribe

This topic is the last in a list of tasks that tells you how to grant access to publish and subscribe to topics by more than one user.

Before you begin

This topic uses the setup described in "Grant access to a user to publish to a topic deeper within the tree" on page 837.

About this task

In a previous task `USER1` was given access to subscribe to the topic `"Price/Fruit"`. This topic tells you how to grant access to that user to publish to that topic.

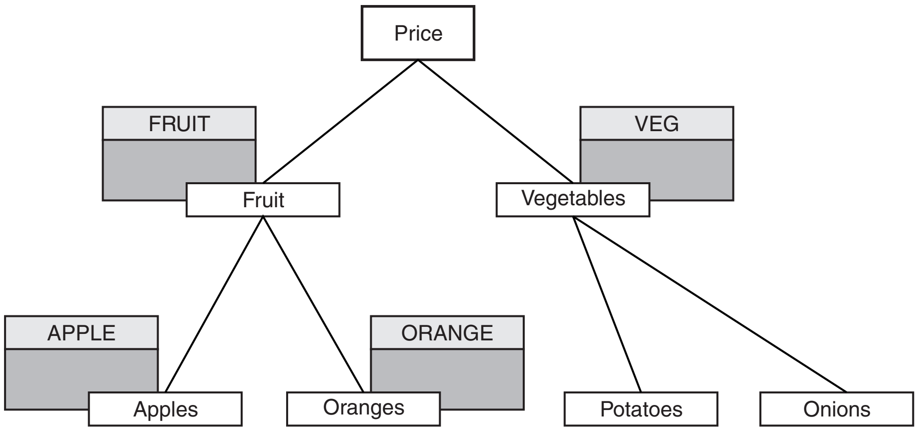


Figure 82. Granting access for publishing and subscribing

Table 91. Example publishing and subscribing access requirements

Topic	Subscribe access required	Publish access required	Topic object
Price	No user	No user	None
Price/Fruit	USER1	USER1	FRUIT
Price/Fruit/Apples	USER1 and USER2		APPLE
Price/Fruit/Oranges	USER1		ORANGE

Procedure

Grant access as follows:

- z/OS **z/OS** :

  In an earlier task USER1 was granted access to subscribe to topic "Price/Fruit" by granting the user access to the hlq.SUBSCRIBE.FRUIT profile.

  In order to publish to the "Price/Fruit" topic, grant access to USER1 to the hlq.PUBLISH.FRUIT profile. Do this, using the following RACF commands:

  ```
  RDEFINE MXTOPIC hlq.PUBLISH.FRUIT UACC(NONE)
  PERMIT hlq.PUBLISH.FRUIT CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
  ```

- Other platforms:

  Grant access to USER1 to publish to topic "Price/Fruit" by granting the user publish access to the FRUIT profile. Do this, using the authorization command for the platform:

  Windows UNIX Linux **Windows, UNIX and Linux systems**

  ```
  setmqaut -t topic -n FRUIT -p USER1 +pub
  ```

  IBM i **IBM i**

  ```
  GRTMQAUT OBJ(FRUIT) OBJTYPE(*TOPIC) USER(USER1) AUT(*PUB)
  ```

Results

On z/OS, when USER1 attempts to publish to topic "Price/Fruit" the security check on the MQOPEN call passes.

When USER2 attempts to publish at topic "Price/Fruit" the result is failure with an MQRC_NOT_AUTHORIZED message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

  ```
  ICH408I USER(USER2   ) ...
    hlq.PUBLISH.FRUIT ...

  ICH408I USER(USER2   ) ...
    hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
  ```

- Windows UNIX Linux On Windows, UNIX, and Linux platforms, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_OPEN_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
  TopicString     "Price/Fruit"
  ```

- IBM i On IBMi, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier   MQRQ_OPEN_NOT_AUTHORIZED
  UserIdentifier     USER2
  AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
  TopicString     "Price/Fruit"
  ```

Following the complete set of these tasks, gives USER1 and USER2 the following access authorities for publish and subscribe to the topics listed:

Table 92. Complete list of access authorities resulting from security examples

Topic	Subscribe access required	Publish access required	Topic object
Price	No user	No user	None
Price/Fruit	USER1	USER1	FRUIT
Price/Fruit/Apples	USER1 and USER2		APPLE
Price/Fruit/Oranges	USER1		ORANGE
Price/Vegetables		USER1	VEG
Price/Vegetables/Potatoes			
Price/Vegetables/Onions			

Where you have different requirements for security access at different levels within the topic tree, careful planning ensures that you do not receive extraneous security warnings on the z/OS console log. Setting up security at the correct level within the tree avoids misleading security messages.

Subscription security

MQSO_ALTERNATE_USER_AUTHORITY

The AlternateUserId field contains a user identifier to use to validate this MQSUB call. The call can succeed only if this AlternateUserId is authorized to subscribe to the topic with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

MQSO_SET_IDENTITY_CONTEXT

The subscription is to use the accounting token and application identity data supplied in the PubAccountingToken and PubApplIdentityData fields.

If this option is specified, the same authorization check is carried out as if the destination queue was accessed using an MQOPEN call with MQOO_SET_IDENTITY_CONTEXT, except in the case where the MQSO_MANAGED option is also used in which case there is no authorization check on the destination queue.

If this option is not specified, the publications sent to this subscriber have default context information associated with them as follows:

Table 93. Default publication context information

Field in MQMD	Value used
UserIdentifier	The user ID associated with the subscription (see SUBUSER field on DISPLAY SBSTATUS) at the time the publication is made.
AccountingToken	Determined from the environment if possible; set to MQACT_NONE otherwise.
ApplIdentityData	Set to blanks.

This option is only valid with MQSO_CREATE and MQSO_ALTER. If used with MQSO_RESUME, the PubAccountingToken and PubApplIdentityData fields are ignored, so this option has no effect.

If a subscription is altered without using this option where previously the subscription had supplied identity context information, default context information is generated for the altered subscription.

If a subscription allowing different user IDs to use it with option MQSO_ANY_USERID, is resumed by a different user ID, default identity context is generated for the new user ID now owning the subscription and any subsequent publications are delivered containing the new identity context.

AlternateSecurityId

This is a security identifier that is passed with the AlternateUserId to the authorization service to allow appropriate authorization checks to be performed. AlternateSecurityId is used only if MQSO_ALTERNATE_USER_AUTHORITY is specified, and the AlternateUserId field is not entirely blank up to the first null character or the end of the field.

MQSO_ANY_USERID subscription option

When MQSO_ANY_USERID is specified, the identity of the subscriber is not restricted to a single user ID. This allows any user to alter or resume the subscription when they have suitable authority. Only a single user may have the subscription at any one time. An attempt to resume use of a subscription currently in use by another application will cause the call to fail with MQRC_SUBSCRIPTION_IN_USE.

To add this option to an existing subscription the MQSUB call (using MQSO_ALTER) must come from the same user ID as the original subscription.

If an MQSUB call refers to an existing subscription with MQSO_ANY_USERID set, and the user ID differs from the original subscription, the call succeeds only if the new user ID has authority to subscribe to the topic. After successful completion, future publications to this subscriber are put to the subscriber's queue with the new user ID set in the publication.

MQSO_FIXED_USERID

When MQSO_FIXED_USERID is specified, the subscription can only be altered or resumed by a single owning user ID. This user ID is the last user ID to alter the subscription that set this option, thereby removing the MQSO_ANY_USERID option, or if no alters have taken place, it is the user ID that created the subscription.

If an MQSUB verb refers to an existing subscription with MQSO_ANY_USERID set and alters the subscription (using MQSO_ALTER) to use option MQSO_FIXED_USERID, the user ID of the subscription is now fixed at this new user ID. The call succeeds only if the new user ID has authority to subscribe to the topic.

If a user ID other than the one recorded as owning a subscription trys to resume or alter an MQSO_FIXED_USERID subscription, the call will fail with MQRC_IDENTITY_MISMATCH. The owning user ID of a subscription can be viewed using the DISPLAY SBSTATUS command.

If neither MQSO_ANY_USERID or MQSO_FIXED_USERID is specified, the default is MQSO_FIXED_USERID.

Publish/subscribe security between queue managers

Publish/subscribe internal messages, such as proxy subscriptions and publications, are put to publish/subscribe system queues using normal channel security rules. The information and diagrams in this topic highlight the various processes and user IDs involved in the delivery of these messages.

Local access control

Access to topics for publication and subscriptions is governed by local security definitions and rules that are described in Publish/subscribe security. On z/OS, no local topic object is required to establish access control. No local topic is required for access control on other platforms either. Administrators can choose to apply access control to clustered topic objects, irrespective of whether they exist in the cluster yet.

System administrators are responsible for access control on their local system. They must trust the administrators of other members of the hierarchy or cluster collectives to be responsible for their access control policy. Because access control is defined for each separate machine it is likely to be burdensome if fine level control is needed. It might not be necessary to impose any access control, or access control might be defined on high-level objects in the topic tree. Fine level access control can be defined for each subdivision of the topic namespace.

Making a proxy subscription

Trust for an organization to connect its queue manager to your queue manager is confirmed by normal channel authentication means. If that trusted organization is also allowed to do distributed publish/subscribe, an authority check is done. The check is made when the channel puts a message to a distributed publish/subscribe queue. For example, if a message is put to the `SYSTEM.INTER.QMGR.CONTROL` queue. The user ID for the queue authority check depends on the `PUTAUT` values of the receiving channel. For example, the user ID of the channel, `MCAUSER`, the message context, depending on the value and platform. For more information about channel security, see Channel security.

Proxy subscriptions are made with the user ID of the distributed publish/subscribe agent on the remote queue manager. For example, `QM2` in Figure 83. The user is then easily granted access to local topic object profiles, because that user ID is defined in the system and there are therefore no domain conflicts.

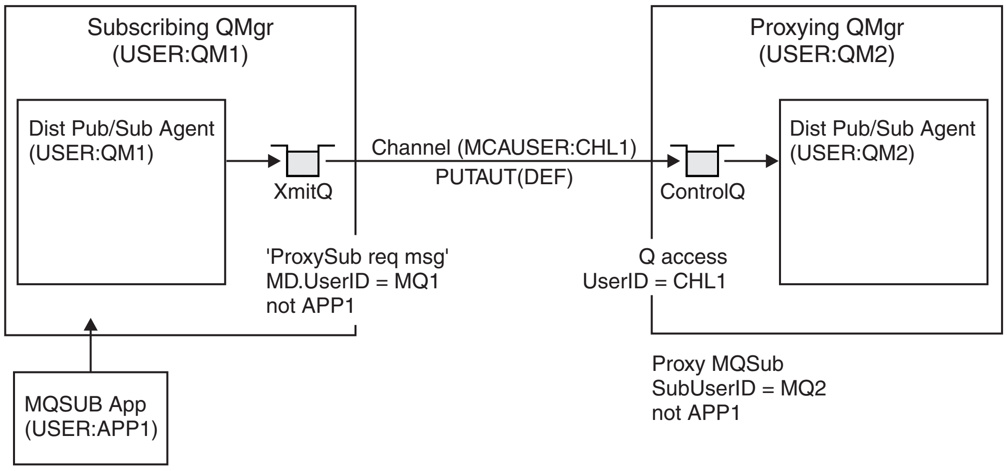


Figure 83. Proxy subscription security, making a subscription

Sending back remote publications

When a publication is created on the publishing queue manager, a copy of the publication is created for any proxy subscription. The context of the copied publication contains the context of the user ID which made the subscription; QM2 in Figure 84. The proxy subscription is created with a destination queue that is a remote queue, so the publication message is resolved onto a transmission queue.

Trust for an organization to connect its queue manager, QM2, to another queue manager, QM1, is confirmed by normal channel authentication means. If that trusted organization is then allowed to do distributed publish/subscribe, an authority check is done when the channel puts the publication message to the distributed publish/subscribe publication queue SYSTEM.INTER.QMGR.PUBS. The user ID for the queue authority check depends on the PUTAUT value of the receiving channel (for example, the user ID of the channel, MCAUSER, message context, and others, depending on value and platform). For more information about channel security, see Channel security.

When the publication message reaches the subscribing queue manager, another MQPUT to the topic is done under the authority of that queue manager and the context with the message is replaced by the context of each of the local subscribers as they are each given the message.

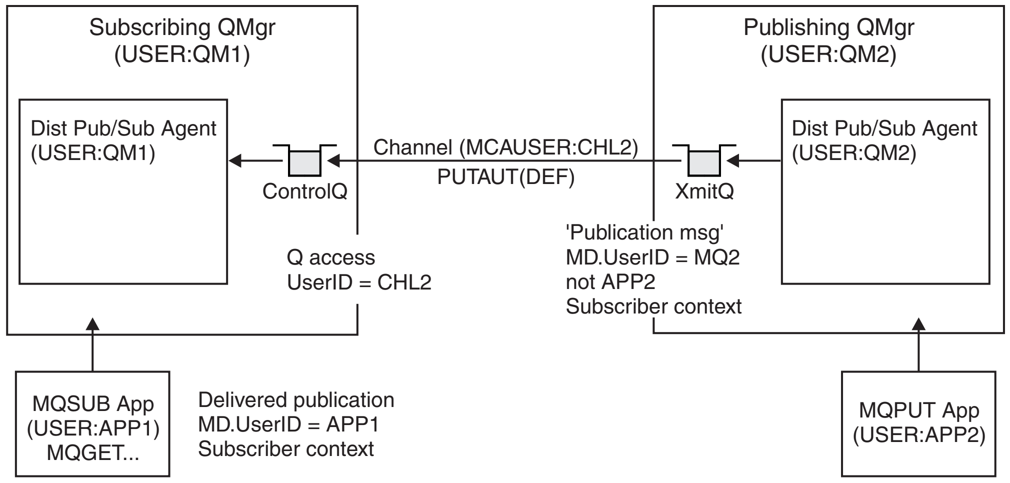


Figure 84. Proxy subscription security, forwarding publications

On a system where little has been considered regarding security, the distributed publish/subscribe processes are likely to be running under a user ID in the mqm group, the MCAUSER parameter on a channel is blank (the default), and messages are delivered to the various system queues as required. The unsecured system makes it easy to set up a proof of concept to demonstrate distributed publish/subscribe.

On a system where security is more seriously considered, these internal messages are subject to the same security controls as any message going over the channel.

If the channel is set up with a non-blank MCAUSER and a PUTAUT value specifying that MCAUSER must be checked, then the MCAUSER in question must be granted access to SYSTEM.INTER.QMGR.* queues. If there are multiple different remote queue managers, with channels running under different MCAUSER IDs, all those user IDs need to be granted access to the SYSTEM.INTER.QMGR.* queues. Channels running under different MCAUSER IDs might occur, for example, when multiple hierarchical connections are configured on a single queue manager.

If the channel is set up with a PUTAUT value specifying that the context of the message is used, then access to the SYSTEM.INTER.QMGR.* queues are checked based on the user ID inside the internal message. Because all these messages are put with the user ID of the distributed publish/subscribe agent from the queue manager that is sending the internal message, or publication message (see Figure 84 on page 843), it is not too large a set of user IDs to grant access to the various system queues (one per remote queue manager), if you want to set up your distributed publish/subscribe security in this way. It still has all the same issues that channel context security always has; that of the different user ID domains and the fact that the user ID in the message might not be defined on the receiving system. However, it is a perfectly acceptable way to run if required.

z/OS System queue security provides a list of queues and the access that is required to securely set up your distributed publish/subscribe environment. If any internal messages or publications fail to be put due to security violations, the channel writes a message to the log in the normal manner and the messages can be sent to the dead-letter queue according to normal channel error processing.

All inter-queue manager messaging for the purposes of distributed publish/subscribe runs using normal channel security.

For information about restricting publications and proxy subscriptions at the topic level, see Publish/subscribe security.

Using default user IDs with a queue manager hierarchy

If you have a hierarchy of queue managers running on different platforms and are using default user IDs, note that these default user IDs differ between platforms and might not be known on the target platform. As a result, a queue manager running on one platform rejects messages received from queue managers on other platforms with the reason code `MQRC_NOT_AUTHORIZED`.

To avoid messages being rejected, at a minimum, the following authorities need to be added to the default user IDs used on other platforms:

- *PUT *GET authority on the SYSTEM.BROKER. queues
- *PUB *SUB authority on the SYSTEM.BROKER. topics
- *ADMCRT *ADMDLT *ADMCHG authority on the SYSTEM.BROKER.CONTROL.QUEUE queue.

The default user IDs are as follows:

Platform	**Default user ID**
Windows	`MUSR_MQADMIN` **Note:** `MUSR_MQADMIN` is the default user ID for the first installation only. For subsequent installations, the Prepare IBM MQ Wizard creates a user account named for `MUSR_MQADMINx`, where x is the next available number representing a user ID that does not exist.
UNIX and Linux systems	`mqm`
IBM i	`QMQM`
z/OS	The channel initiator address space user ID

Create and grant access to the 'qmqm' user ID if hierarchically attached to a queue manager on IBM i for Queue Managers on Windows, UNIX, Linux, and z/OS platforms.

Create and grant access to the 'mqm' user ID if hierarchically attached to a queue manager on Windows, UNIX, or Linux for Queue Managers on IBM i and z/OS platforms.

Create and grant user access to the z/OS channel initiator address space user ID if hierarchically attached to a queue manager on z/OS for Queue Managers on Windows, UNIX, Linux, and IBM i platforms.

User IDs can be case sensitive. The originating queue manager (if IBM i, Windows, UNIX, or Linux systems) forces the user ID to be all uppercase. The receiving queue manager (if Windows, UNIX or Linux systems) forces the user ID to be all lowercase. Therefore, all user IDs created on UNIX and Linux systems must be created in their lowercase form. If a message exit has been installed, forcing the user ID into uppercase or lowercase does not take place. Care must be taken to understand how the message exit processes the user ID.

To avoid potential problems with conversion of user IDs:
- On UNIX, Linux and Windows systems, ensure the user IDs are specified in lowercase.
- On IBM i and z/OS, ensure the user IDs are specified in uppercase.

Protection of database authentication details

V 8.0.0.4

If your are using user name and password authentication to connect to the database manager, you can store them in the MQ XA credentials store to avoid storing the password in plain text in the `qm.ini` file.

Update XAOpenString for the resource manager

To use the credentials store you must modify XAOpenString in the `qm.ini` file. The string is used to connect to the database manager. You specify replaceable fields to identify where the user name and password are substituted within the XAOpenString string.

- The `+USER+` field is replaced with the user name value stored in the XACredentials store.
- The `+PASSWORD+` field is replaced with the password value stored in the XACredentials store.

The following examples show how to modify an XAOpenString to use the credentials file to connect to the database.

Connecting to a Db2 database

```
XAResourceManager:
  Name=mydb2
  SwitchFile=db2swit
  XAOpenString=db=mydbname,uid=+USER+,pwd=+PASSWORD+,toc=t
  ThreadOfControl=THREAD
```

Connecting to an Oracle database

```
XAResourceManager:
  Name=myoracle
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/+USER+/+PASSWORD++SesTm=35
            +LogDir=/tmp+threads=true
  ThreadOfControl=THREAD
```

Work with the credentials for the database to the MQ XA credentials store

After you update the `qm.ini` file with the replaceable credential strings, you must add the user name and password to the MQ credentials store by using the **`setmqxacred`** command. You can also use **`setmqxacred`** to modify existing credentials, delete credentials, or list credentials. The following examples give some typical use cases:

Adding credentials

The following command securely saves the user name and password for the queue manager `QM1` for the resource `mqdb2`.

```
setmqxacred -m QM1 -x mydb2 -u user1 -p Password2
```

Updating credentials

To update the user name and password used to connect to a database, re-issue the **`setmqxacred`** command with the new user-name and password:

```
setmqxacred -m QM1 -x mydb2 -u user3 -p Password4
```

You must restart the queue manager for the changes to take effect.

Deleting credentials

The following command deletes the credentials:

```
setmqxacred -m QM1 -x mydb2 -d
```

Listing credentials

The following command lists credentials:

```
setmqxacred -m QM1 -l
```

Related information:

`setmqxacred`

IBM MQ Advanced Message Security

IBM MQ Advanced Message Security (IBM MQ AMS) is a separately licensed component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

IBM MQ AMS overview

IBM MQ applications can use IBM MQ Advanced Message Security to send sensitive data, such as high-value financial transactions and personal information, with different levels of protection by using a public key cryptography model.

Behavior that has changed since Version 7.0.1

IBM MQ Advanced Message Security became a component in IBM MQ from Version 7.5. Some aspects of IBM MQ AMS functionality have changed, that might affect applications, administrative scripts, or management procedures.

Review the following list of changes carefully before upgrading queue managers to Version 7.5 or later. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems:

- IBM MQ AMS installation is a part of the IBM MQ installation process.
- IBM MQ AMS security capability are enabled with its installation and controlled with security policies. You do not need to enable interceptors to allow IBM MQ AMS start intercepting data.
- IBM MQ AMS in IBM MQ does not require the use of the **cfgmqs** command as in the stand-alone version of IBM WebSphere MQ Advanced Message Security.

Features and functions of IBM MQ Advanced Message Security

IBM MQ Advanced Message Security expands IBM MQ security services to provide data signing and encryption at the message level. The expanded services guarantee that message data has not been modified between when it is originally placed on a queue and when it is retrieved. In addition, IBM MQ AMS verifies that a sender of message data is authorized to place signed messages on a target queue.

IBM MQ AMS provides the following functions:

- Secures sensitive or high-value transactions processed by IBM MQ.
- Detects and removes rogue or unauthorized messages before they are processed by a receiving application.
- Verifies that messages were not modified while in transit from queue to queue.
- Protects the data not only as it flows across the network but also when it is put on a queue.
- Secures existing proprietary and customer-written applications for IBM MQ.

Error handling

IBM MQ Advanced Message Security defines an error handling queue to manage messages that contain errors or messages that cannot be unprotected.

Defective messages are dealt with as exceptional cases. If a received message does not meet the security requirements for the queue it is on, for example, if the message is signed when it should be encrypted, or decryption or signature verification fails, the message is sent to the error handling queue. A message might be sent to the error handling queue for the following reasons:

- Quality of protection mismatch - a quality of protection (QOP) mismatch exists between the received message and the QOP definition in the security policy.
- Decryption error - the message cannot be decrypted.
- PDMQ header error - the IBM MQ AMS message header cannot be accessed.
- Size mismatch - length of a message after decryption is different than expected.
- Encryption algorithm strength mismatch - the message encryption algorithm is weaker than required.
- Unknown error - unexpected error occurred.

IBM MQ Advanced Message Security uses the SYSTEM.PROTECTION.ERROR.QUEUE as its error handling queue. All messages put by IBM MQ AMS to the SYSTEM.PROTECTION.ERROR.QUEUE are preceded by an MQDLH header.

Your IBM MQ administrator can also define the SYSTEM.PROTECTION.ERROR.QUEUE as an alias queue pointing to another queue.

Key concepts

Learn about the key concepts in IBM MQ Advanced Message Security to understand how the tool works and how to manage it effectively.

Public key infrastructure

Public key infrastructure (PKI) is a system of facilities, policies, and services that support the use of public key cryptography to obtain secure communication.

There is no single standard that defines the components of a public key infrastructure, but a PKI typically involves usage of public key certificates and comprises certificate authorities (CA) and other registration authorities (RA) that provide the following services:

- Issuing digital certificates
- Validating digital certificates
- Revoking digital certificates
- Distributing certificates

Identity of users and applications are represented by the **distinguished name (DN)** field in a certificate associated with signed or encrypted messages. IBM MQ Advanced Message Security uses this identity to represent a user or an application. To authenticate this identity, the user or application must have access to the keystore where the certificate and associated private key are stored. Each certificate is represented by a label in the keystore.

Related concepts:

"Using keystores and certificates" on page 874
To provide transparent cryptographic protection to IBM MQ applications, IBM MQ Advanced Message Security uses the keystore file, where public key certificates and a private key are stored. On z/OS, a SAF key ring is used instead of a keystore file.

Digital certificates

IBM MQ Advanced Message Security associates users and applications with X.509 standard digital certificates. X.509 certificates are typically signed by a trusted certificate authority (CA) and involve private and public keys which are used for encryption and decryption.

Digital certificates provide protection against impersonation by binding a public key to its owner, whether that owner is an individual, a queue manager, or some other entity. Digital certificates are also known as public key certificates, because they give you assurance about the ownership of a public key when you use an asymmetric key scheme. This scheme requires that a public key and a private key be generated for an application. Data encrypted with the public key can only be decrypted using the corresponding private key while data encrypted with the private key can only be decrypted using the corresponding public key. The private key is stored in a key database file that is password-protected. Only its owner has the access to the private key used to decrypt messages that are encrypted using the corresponding public key.

If public keys are sent directly by their owner to another entity, there is a risk that the message could be intercepted and the public key substituted by another. This is known as a "man-in-the-middle" attack. The solution is to exchange public keys through a trusted third party, giving the user a strong assurance that the public key belongs to the entity with which you are communicating. Instead of sending your public key directly, you ask a trusted third party to incorporate it into a digital certificate. The trusted third-party who issues digital certificates is called a certificate authority (CA).

For more information about digital certificates, see What is in a digital certificate.

A digital certificate contains the public key for an entity and states that the public key belongs to that entity:

- when a certificate is for an individual entity, it is called a *personal certificate* or *user certificate*.
- when a certificate is for a certificate authority, the certificate is called a *CA certificate* or *signer certificate*.

Note: IBM MQ Advanced Message Security supports self-signed certificates in both Java and native applications

Related information:

Cryptography
Cryptography is the process of converting between readable text, called *plaintext*, and an unreadable form, called *ciphertext*.

Object authority manager

On platforms other than z/OS, the Object Authority Manager (OAM) is the authorization service component supplied with the IBM MQ products.

The access to IBM MQ Advanced Message Security entities is controlled through IBM MQ user groups and the OAM. Administrators can use the command-line interface to grant or revoke authorizations as required. Different groups of users can have different kinds of access authority to the same objects. For example, one group could perform both PUT and GET operations for a specific queue while another group might be allowed only to browse the queue. Similarly, some groups might have GET and PUT authority to a queue, but are not allowed to alter or delete the queue.

Through the OAM, you can control:

- Access to IBM MQ Advanced Message Security objects through MQI. When an application program attempts to access objects, the OAM checks if the user profile making the request has the authorization for the operation requested. This means that queues, and the messages on queues, can be protected from unauthorized access.
- Permission to use PCF and MQSC commands.

Related information:

Object authority manager

Supported technology

IBM MQ Advanced Message Security depends on several technology components to provide a security infrastructure.

IBM MQ Advanced Message Security supports the following IBM MQ application programming interfaces (APIs):

- Message queue interface (MQI)
- IBM MQ Java Message Service (JMS) 1.0.2 and 1.1.
- IBM MQ Base Classes for Java
- IBM MQ classes for .Net in an unmanaged mode

Note: IBM MQ Advanced Message Security supports X.509 compliant certificate authorities.

Known limitations

Learn about limitations of IBM MQ Advanced Message Security.

- The following IBM MQ options are not supported:
  - Publish/subscribe.
  - Channel data conversion.
  - Distribution lists.
  - Application message segmentation
  - The use of non-threaded applications using API exit on HP-UX platforms.
  - IBM MQ classes for .NET in a managed mode (client or bindings connections).
  - Message Service client for .NET (XMS) applications.
  - Message Service client for C/C++ (XMS supportPac IA94) applications.
  - IBM MQ queues processed by the IMS bridge.

  Note: IBM MQ AMS is supported on CICS Bridge queues. You should use the same user ID to MQPUT (encrypt) and MQGET (decrypt) on CICS Bridge queues.
- All Java applications are dependant on IBM Java Runtime. IBM MQ AMS does not support any JRE provided by other vendors.
- Users should avoid putting two or more certificates with the same Distinguished Names in a single keystore file because the IBM MQ AMS interceptor's functioning with such certificates is undefined.
- IBM MQ AMS is not supported in JMS if the **`WMQ_PROVIDER_VERSION`** property is set to 6.

User scenarios

Familiarize yourself with possible scenarios to understand what business goals you can achieve with IBM MQ Advanced Message Security.

Quick Start Guide for IBM MQ AMS on Windows platforms

Use this guide to quickly configure IBM MQ Advanced Message Security to provide message security on Windows platforms. By the time you complete it, you will have created a key database to verify user identities, and defined signing/encryption policies for your queue manager.

Before you begin

You should have at least the following features installed on your system:
- Server
- Development Toolkit (for the Sample programs)
- Advanced Message Security

Refer to IBM MQ features for Windows systems for details.

1. Creating a queue manager and a queue:
About this task

All the following examples use a queue named `TEST.Q` for passing messages between applications. IBM MQ Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the IBM MQ infrastructure through the standard IBM MQ interface. The basic setup is done in IBM MQ and is configured in the following steps.

You can use IBM MQ Explorer to create the queue manager QM_VERIFY_AMS and its local queue called `TEST.Q` by using all the default wizard settings, or you can use the commands found in `C:\Program Files\IBM\WebSphere MQ\bin`. Remember that you must be a member of the `mqm` user group to run the following administrative commands.

Procedure

1. Create a queue manager
   ```
   crtmqm QM_VERIFY_AMS
   ```
2. Start the queue manager
   ```
   strmqm QM_VERIFY_AMS
   ```
3. Create a queue called `TEST.Q` by entering the following command into **runmqsc** for queue manager `QM_VERIFY_AMS`
   ```
   DEFINE QLOCAL(TEST.Q)
   ```

Results

If the procedure is completed, command entered into **runmqsc** will display details about `TEST.Q`:

```
DISPLAY Q(TEST.Q)
```

2. Creating and authorizing users:
About this task

There are two users that appear in this example: `alice`, the sender, and `bob`, the receiver. To use the application queue, these users need to be granted authority to use it. Also to successfully use the protection policies that we will define these users must be granted access to some system queues. For more information about the **setmqaut** command refer to **setmqaut** .

Procedure

1. Create the two users and ensure that HOMEPATH and HOMEDRIVE are set for both these users.
2. Authorize the users to connect to the queue manager and to work with the queue

   ```
   setmqaut -m QM_VERIFY_AMS -t qmgr -p alice -p bob +connect +inq
   setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p alice +put
   setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p bob +get
   ```

3. You must also allow the two users to browse the system policy queue and put messages on the error queue.

   ```
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p alice -p bob +browse
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p alice -p bob +put
   ```

Results

Users are now created and the required authorities granted to them.

What to do next

To verify if the steps were carried out correctly, use the amqsput and amqsget samples as described in section “7. Testing the setup” on page 856.

3. Creating key database and certificates:
About this task

Interceptor requires the public key of the sending users to encrypt the message. Thus, the key database of user identities mapped to public and private keys must be created. In the real system, where users and applications are dispersed over several computers, each user would have its own private keystore. Similarly, in this guide, we create key databases for alice and bob and share the user certificates between them.

Note: In this guide, we use sample applications written in C connecting using local bindings. If you plan to use Java applications using client bindings, you must create a JKS keystore and certificates using the **keytool** command, which is part of the JRE (see “Quick Start Guide for IBM MQ AMS with Java clients” on page 863 for more details). For all other languages, and for Java applications using local bindings, the steps in this guide are correct.

Procedure

1. Use the IBM Key Management GUI (strmqikm.exe) to create a new key database for the user alice.

   ```
   Type:       CMS
   Filename:   alicekey.kdb
   Location:   C:/Documents and Settings/alice/AMS
   ```

   Note:
   - It is advisable to use a strong password to secure the database.
   - Make sure that **Stash password to a file** check box is selected.
2. Change the key database content view to **Personal Certificates**.
3. Select **New Self Signed** ; self signed certificates are used in this scenario.
4. Create a certificate identifying the user alice for use in encryption, using these fields:

   ```
   Key label: Alice_Cert
   Common Name: alice
   Organisation: IBM
   Country: GB
   ```

   Note:

- For the purpose of this guide, we are using self-signed certificate which can be created without using a Certificate Authority. For production systems, it is advisable not to use self-signed certificates but instead rely on certificates signed by a Certificate Authority.
- The **`Key label`** parameter specifies the name for the certificate, which interceptors will look up to receive necessary information.
- The **`Common Name`** and optional parameters specifies the details of the **Distinguished Name** (DN), which must be unique for each user.

5. Repeat step 1-4 for the user `bob`

Results

The two users `alice` and `bob` each now have a self-signed certificate.

4. Creating keystore.conf:
About this task

You must point IBM MQ Advanced Message Security interceptors to the directory where the key databases and certificates are located.This is done via the `keystore.conf` file, which hold that information in the plain text form. Each user must have a separate `keystore.conf` file in the `.mqs` folder. This step must be done for both, `alice` and `bob`.

The content of `keystore.conf` should be of the form:

```
cms.keystore = <dir>/keystore_file
cms.certificate = certificate_label
```

Example

For this scenario, the contents of the `keystore.conf` will be as follows:

```
cms.keystore = C:/Documents and Settings/alice/AMS/alicekey
cms.certificate = Alice_Cert
```

Note:
- The path to the keystore file must be provided with no file extension.
- The certificate label can include spaces, thus "Alice_Cert" and "Alice_Cert " (with a space on the end) for example, are recognized as labels of two different certificates. However, to avoid confusion, it is better not to use spaces in label's name.
- There are the following keystore formats: `CMS` (Cryptographic Message Syntax), `JKS` (Java Keystore) and `JCEKS` (Java Cryptographic Extension Keystore). For more information, refer to "Structure of the configuration file" on page 875.
- *`%HOMEDRIVE%\%HOMEPATH%`*`\.mqs\keystore.conf` (eg. C:\Documents and Settings\alice\.mqs\keystore.conf) is the default location where IBM MQ Advanced Message Security searches for the `keystore.conf` file. For information about how to use a non-default location for the `keystore.conf`, see "Using keystores and certificates" on page 874.
- To create `.mqs` directory, you must use the command prompt.

5. Sharing Certificates:
About this task

Share the certificates between the two key databases so that each user can successfully identify the other. This is done by extracting each user's public certificate to a file, which is then added to the other user's key database.

Note: Take care to use the *extract* option, and not the *export* option. *Extract* gets the user's public key, whereas *export* gets both the public and private key. Using *export* by mistake would completely compromise your application, by passing on its private key.

Procedure

1. Extract the certificate identifying `alice` to an external file:
   ```
   runmqakm -cert -extract -db "C:/Documents and Settings/alice/AMS/alicekey.kdb" -pw passw0rd -label Alice_Cert -target al
   ```
2. Add the certificate to `bob's` keystore:
   ```
   runmqakm -cert -add -db "C:/Documents and Settings/bob/AMS/bobkey.kdb" -pw passw0rd -label Alice_Cert -file alice_public
   ```
3. Repeat steps for `bob`:
   ```
   runmqakm -cert -extract -db "C:/Documents and Settings/bob/AMS/bobkey.kdb" -pw passw0rd -label Bob_Cert -target bob_publ
   ```
   ```
   runmqakm -cert -add -db "C:/Documents and Settings/alice/AMS/alicekey.kdb" -pw passw0rd -label Bob_Cert -file bob_public
   ```

Results

The two users `alice` and `bob` are now able to successfully identify each other having created and shared self-signed certificates.

What to do next

Verify that a certificate is in the keystore either by browsing it using the GUI or running the following commands which print out its details:

```
runmqakm -cert -details -db "C:/Documents and Settings/bob/AMS/bobkey.kdb" -pw passw0rd -label Alice_Cert
runmqakm -cert -details -db "C:/Documents and Settings/alice/AMS/alicekey.kdb" -pw passw0rd -label Bob_Cert
```

6. Defining queue policy:
About this task

With the queue manager created and interceptors prepared to intercept messages and access encryption keys, we can start defining protection policies on `QM_VERIFY_AMS` using the `setmqspl` command. Refer to setmqspl for more information on this command. Each policy name must be the same as the queue name it is to be applied to.

Example

This is an example of a policy defined for the `TEST.Q` queue. In the example, messages are signed with the `SHA1` algorithm and encrypted with the `AES256` algorithm. `alice` is the only valid sender and `bob` is the only receiver of the messages on this queue:

```
setmqspl -m QM_VERIFY_AMS -p TEST.Q -s SHA1 -a "CN=alice,O=IBM,C=GB" -e AES256 -r "CN=bob,O=IBM,C=GB"
```

Note: The DNs match exactly those specified in the receptive user's certificate from the key database.

What to do next

To verify the policy you have defined, issue the following command:

```
dspmqspl -m QM_VERIFY_AMS
```

To print the policy details as a set of `setmqspl` commands, the `-export` flag. This allows storing already defined policies:

```
dspmqspl -m QM_VERIFY_AMS -export >restore_my_policies.bat
```

7. Testing the setup:

About this task

By running different programs under different users you can verify if the application has been properly configured.

Procedure

1. Switch user to run as user `alice`

   Right-click `cmd.exe` and select **Run as...**. When prompted, log in as the user `alice`.
2. As the user `alice` put a message using a sample application:
   ```
   amqsput TEST.Q QM_VERIFY_AMS
   ```
3. Type the text of the message, then press Enter.
4. Switch user to run as user `bob`

   Open another window by right-clicking `cmd.exe` and selecting **Run as...**. When prompted, log in as the user `bob`.
5. As the user `bob` get a message using a sample application:
   ```
   amqsget TEST.Q QM_VERIFY_AMS
   ```

Results

If the application has been configured properly for both users, the user `alice` 's message is displayed when `bob` runs the getting application.

8. Testing encryption:
About this task

To verify that the encryption is occurring as expected, create an alias queue which references the original queue `TEST.Q`. This alias queue will have no security policy and so no user will have the information to decrypt the message and therefore the encrypted data will be shown.

Procedure

1. Using the **runmqsc** command against queue manager QM_VERIFY_AMS, create an alias queue.
   ```
   DEFINE QALIAS(TEST.ALIAS) TARGET(TEST.Q)
   ```
2. Grant `bob` access to browse from the alias queue
   ```
   setmqaut -m QM_VERIFY_AMS -n TEST.ALIAS -t queue -p bob +browse
   ```
3. As the user `alice`, put another message using a sample application just as before:
   ```
   amqsput TEST.Q QM_VERIFY_AMS
   ```
4. As the user `bob`, browse the message using a sample application via the alias queue this time:
   ```
   amqsbcg TEST.ALIAS QM_VERIFY_AMS
   ```
5. As the user `bob`, get the message using a sample application from the local queue:
   ```
   amqsget TEST.Q QM_VERIFY_AMS
   ```

Results

The output from the `amqsbcg` application shows the encrypted data that is on the queue proving that the message has been encrypted.

Quick Start Guide for IBM MQ AMS on UNIX platforms

Use this guide to quickly configure IBM MQ Advanced Message Security to provide message security on UNIX platforms. By the time you complete it, you will have created a key database to verify user identities, and defined signing/encryption policies for your queue manager.

Before you begin

You should have at least the following components installed on your system:
- Runtime
- Server
- Sample programs
- IBM Global Security Kit
- MQ Advanced Message Security

Refer to the following topics for the component names on each specific platform:
- IBM MQ components for Linux systems
- IBM MQ components for HP-UX systems
- IBM MQ components for AIX systems
- IBM MQ components for Solaris systems

1. Creating a queue manager and a queue:
About this task

All the following examples use a queue named `TEST.Q` for passing messages between applications. IBM MQ Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the IBM MQ infrastructure through the standard IBM MQ interface. The basic setup is done in IBM MQ and is configured in the following steps.

You can use IBM MQ Explorer to create the queue manager QM_VERIFY_AMS and its local queue called `TEST.Q` by using all the default wizard settings, or you can use the commands found in `<MQ_INSTALL_PATH>/bin`. Remember that you must be a member of the `mqm` user group to run the following administrative commands.

Procedure

1. Create a queue manager
   ```
   crtmqm QM_VERIFY_AMS
   ```
2. Start the queue manager
   ```
   strmqm QM_VERIFY_AMS
   ```
3. Create a queue called `TEST.Q` by entering the following command into **runmqsc** for queue manager `QM_VERIFY_AMS`
   ```
   DEFINE QLOCAL(TEST.Q)
   ```

Results

If the procedure completed successfully, the following command entered into **runmqsc** will display details about `TEST.Q`:

```
DISPLAY Q(TEST.Q)
```

2. Creating and authorizing users:
About this task

There are two users that appear in this example: `alice`, the sender, and `bob`, the receiver. To use the application queue, these users need to be granted authority to use it. Also to successfully use the protection policies that we will define these users must be granted access to some system queues. For more information about the **setmqaut** command refer to **setmqaut** .

Procedure

1. Create the two users
   ```
   useradd alice
   useradd bob
   ```
2. Authorize the users to connect to the queue manager and to work with the queue
   ```
   setmqaut -m QM_VERIFY_AMS -t qmgr -p alice -p bob +connect +inq
   setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p alice +put
   setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p bob +get
   ```
3. You must also allow the two users to browse the system policy queue and put messages on the error queue.
   ```
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p alice -p bob +browse
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p alice -p bob +put
   ```

Results

User groups are now created and the required authorities granted to them. This way users who are assigned to those groups will also have permission to connect to the queue manager and to put and get from the queue.

What to do next

To verify if the steps were carried out correctly, use the `amqsput` and `amqsget` samples as described in section "8. Testing encryption" on page 862.

3. Creating key database and certificates:
About this task

To encrypt the message, the interceptor requires the private key of the sending user and the public key(s) of the recipient(s). Thus, the key database of user identities mapped to public and private keys must be created. In the real system, where users and applications are dispersed over several computers, each user would have its own private keystore. Similarly, in this guide, we create key databases for `alice` and `bob` and share the user certificates between them.

Note: In this guide, we use sample applications written in C connecting using local bindings. If you plan to use Java applications using client bindings, you must create a JKS keystore and certificates using the **`keytool`** command, which is part of the JRE (see "Quick Start Guide for IBM MQ AMS with Java clients" on page 863 for more details). For all other languages, and for Java applications using local bindings, the steps in this guide are correct.

Procedure

1. Create a new key database for the user `alice`
   ```
   mkdir /home/alice/.mqs -p
   runmqakm -keydb -create -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -stash
   ```
   Note:
   - It is advisable to use a strong password to secure the database.
   - The `stash` parameter stores the password into the `key.sth` file, which interceptors can use to open the database.
2. Ensure the key database is readable
   ```
   chmod +r /home/alice/.mqs/alicekey.kdb
   ```
3. Create a certificate identifying the user `alice` for use in encryption
   ```
   runmqakm -cert -create -db /home/alice/.mqs/alicekey.kdb -pw passw0rd
   -label Alice_Cert -dn "cn=alice,O=IBM,c=GB" -default_cert yes
   ```

Note:

- For the purpose of this guide, we are using self-signed certificate which can be created without using a Certificate Authority. For production systems, it is advisable not to use self-signed certificates but instead rely on certificates signed by a Certificate Authority.
- The `label` parameter specifies the name for the certificate, which interceptors will look up to receive necessary information.
- The `DN` parameter specifies the details of the **Distinguished Name** (DN), which must be unique for each user.

4. Now we have created the key database, we should set the ownership of it, and ensure it is unreadable by all other users.

   ```
   chown alice /home/alice/.mqs/alicekey.kdb /home/alice/.mqs/alicekey.sth
   chmod 600 /home/alice/.mqs/alicekey.kdb /home/alice/.mqs/alicekey.sth
   ```
5. Repeat step 1-4 for the user `bob`

Results

The two users `alice` and `bob` each now have a self-signed certificate.

4. Creating keystore.conf:
About this task

You must point IBM MQ Advanced Message Security interceptors to the directory where the key databases and certificates are located. This is done via the `keystore.conf` file, which hold that information in the plain text form. Each user must have a separate `keystore.conf` file. This step should be done for both `alice` and `bob`.

The content of `keystore.conf` must be of the form:

```
cms.keystore = <dir>/keystore_file
cms.certificate = certificate_label
```

Example

For this scenario, the contents of the `keystore.conf` will be as follows:

```
cms.keystore = /home/alice/.mqs/alicekey
cms.certificate = Alice_Cert
```

Note:

- The path to the keystore file must be provided with no file extension.
- There are the following keystore formats: `CMS` (Cryptographic Message Syntax), `JKS` (Java Keystore) and `JCEKS` (Java Cryptographic Extension Keystore). For more information, refer to “Structure of the configuration file” on page 875.
- `HOME/.mqs/keystore.conf` is the default location where IBM MQ Advanced Message Security searches for the `keystore.conf` file. For information about how to use a non-default location for the `keystore.conf`, see “Using keystores and certificates” on page 874.

5. Sharing Certificates:
About this task

Share the certificates between the two key databases so that each user can successfully identify the other. This is done by extracting each user's public certificate to a file, which is then added to the other user's key database.

Note: Take care to use the *extract* option, and not the *export* option. *Extract* gets the user's public key, whereas *export* gets both the public and private key. Using *export* by mistake would completely compromise your application, by passing on its private key.

Procedure

1. Extract the certificate identifying `alice` to an external file:
   ```
   runmqakm -cert -extract -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -label Alice_Cert -target alice_public.arm
   ```
2. Add the certificate to `bob's` keystore:
   ```
   runmqakm -cert -add -db /home/bob/.mqs/bobkey.kdb -pw passw0rd -label Alice_Cert -file alice_public.arm
   ```
3. Repeat the step for `bob`:
   ```
   runmqakm -cert -extract -db /home/bob/.mqs/bobkey.kdb -pw passw0rd -label Bob_Cert -target bob_public.arm
   ```
4. Add the certificate for `bob` to `alice's` keystore:
   ```
   runmqakm -cert -add -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -label Bob_Cert -file bob_public.arm
   ```

Results

The two users `alice` and `bob` are now able to successfully identify each other having created and shared self-signed certificates.

What to do next

Verify that a certificate is in the keystore by running the following commands which print out its details:

```
runmqakm -cert -details -db /home/bob/.mqs/bobkey.kdb -pw passw0rd -label Alice_Cert
runmqakm -cert -details -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -label Bob_Cert
```

6. Defining queue policy:
About this task

With the queue manager created and interceptors prepared to intercept messages and access encryption keys, we can start defining protection policies on `QM_VERIFY_AMS` using the `setmqspl` command. Refer to setmqspl for more information on this command. Each policy name must be the same as the queue name it is to be applied to.

Example

This is an example of a policy defined for the `TEST.Q` queue. In this example, messages are signed by the user `alice` using the `SHA1` algorithm, and encrypted using the 256-bit `AES` algorithm. `alice` is the only valid sender and `bob` is the only receiver of the messages on this queue:

```
setmqspl -m QM_VERIFY_AMS -p TEST.Q -s SHA1 -a "CN=alice,O=IBM,C=GB" -e AES256 -r "CN=bob,O=IBM,C=GB"
```

Note: The DNs match exactly those specified in the receptive user's certificate from the key database.

What to do next

To verify the policy you have defined, issue the following command:

```
dspmqspl -m QM_VERIFY_AMS
```

To print the policy details as a set of `setmqspl` commands, the `-export` flag. This allows storing already defined policies:

```
dspmqspl -m QM_VERIFY_AMS -export >restore_my_policies.bat
```

7. Testing the setup:

About this task

By running different programs under different users you can verify if the application has been properly configured.

Procedure

1. Change to the directory containing the samples. If MQ is installed in a non-default location, this may be in a different place.
   ```
   cd /opt/mqm/samp/bin
   ```
2. Switch user to run as user `alice`
   ```
   su alice
   ```
3. As the user `alice`, put a message using a sample application:
   ```
   ./amqsput TEST.Q QM_VERIFY_AMS
   ```
4. Type the text of the message, then press Enter.
5. Stop running as user `alice`
   ```
   exit
   ```
6. Switch user to run as user `bob`
   ```
   su bob
   ```
7. As the user `bob`, get a message using a sample application:
   ```
   ./amqsget TEST.Q QM_VERIFY_AMS
   ```

Results

If the application has been configured properly for both users, the user `alice` 's message is displayed when `bob` runs the getting application.

8. Testing encryption:
About this task

To verify that the encryption is occurring as expected, create an alias queue which references the original queue `TEST.Q`. This alias queue will have no security policy and so no user will have the information to decrypt the message and therefore the encrypted data will be shown.

Procedure

1. Using the **`runmqsc`** command against queue manager QM_VERIFY_AMS, create an alias queue.
   ```
   DEFINE QALIAS(TEST.ALIAS) TARGET(TEST.Q)
   ```
2. Grant `bob` access to browse from the alias queue
   ```
   setmqaut -m QM_VERIFY_AMS -n TEST.ALIAS -t queue -p bob +browse
   ```
3. As the user `alice`, put another message using a sample application just as before:
   ```
   ./amqsput TEST.Q QM_VERIFY_AMS
   ```
4. As the user `bob`, browse the message using a sample application via the alias queue this time:
   ```
   ./amqsbcg TEST.ALIAS QM_VERIFY_AMS
   ```
5. As the user `bob`, get the message using a sample application from the local queue:
   ```
   ./amqsget TEST.Q QM_VERIFY_AMS
   ```

Results

The output from the `amqsbcg` application will show the encrypted data that is on the queue proving that the message has been encrypted.

Quick Start Guide for IBM MQ AMS with Java clients

Use this guide to quickly configure IBM MQ Advanced Message Security to provide message security for Java applications connecting using client bindings. By the time you complete it, you will have created a keystore to verify user identities, and defined signing/encryption policies for your queue manager.

Before you begin

Ensure you have the appropriate components installed as described in the **Quick Start Guide** (Windows or UNIX).

1. Creating a queue manager and a queue:
About this task

All the following examples use a queue named `TEST.Q` for passing messages between applications. IBM MQ Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the IBM MQ infrastructure through the standard IBM MQ interface. The basic setup is done in IBM MQ and is configured in the following steps.

Procedure

1. Create a queue manager
   ```
   crtmqm QM_VERIFY_AMS
   ```
2. Start the queue manager
   ```
   strmqm QM_VERIFY_AMS
   ```
3. Create and start a listener by entering the following commands into **runmqsc** for queue manager `QM_VERIFY_AMS`
   ```
   DEFINE LISTENER(AMS.LSTR) TRPTYPE(TCP) PORT(1414) CONTROL(QMGR)
   START LISTENER(AMS.LSTR)
   ```
4. Create a channel for our applications to connect in through by entering the following command into **runmqsc** for queue manager `QM_VERIFY_AMS`
   ```
   DEFINE CHANNEL(AMS.SVRCONN) CHLTYPE(SVRCONN)
   ```
5. Create a queue called `TEST.Q` by entering the following command into **runmqsc** for queue manager `QM_VERIFY_AMS`
   ```
   DEFINE QLOCAL(TEST.Q)
   ```

Results

If the procedure completed successfully, the following command entered into **runmqsc** displays details about `TEST.Q`:

```
DISPLAY Q(TEST.Q)
```

2. Creating and authorizing users:
About this task

There are two users that appear in this scenario: `alice`, the sender, and `bob`, the receiver. To use the application queue, these users need to be granted authority to use it. Also to successfully use the protection policies defined in this scenario, these users must be granted access to some system queues. For more information about the **setmqaut** command refer to **setmqaut** .

Procedure

1. Create the two users as described in the **Quick Start Guide** (Windows or UNIX) for your platform.
2. Authorize the users to connect to the queue manager and to work with the queue
   ```
   setmqaut -m QM_VERIFY_AMS -t qmgr -p alice -p bob +connect +inq
   setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p alice +put
   setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p bob +get +inq
   ```

3. You must also allow the two users to browse the system policy queue and put messages on the error queue.

   ```
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p alice -p bob +browse
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p alice -p bob +put
   ```

Results

Users are now created and the required authorities granted to them.

What to do next

To verify if the steps were carried out correctly, use the `JmsProducer` and `JmsConsumer` samples as described in section "7. Testing the setup" on page 866.

3. Creating key database and certificates:
About this task

To encrypt the message to interceptor requires the public key of the sending users. Thus, the key database of user identities mapped to public and private keys must be created. In the real system, where users and applications are dispersed over several computer, each user would have its own private keystore. Similarly, in this guide, we create key databases for `alice` and `bob` and share the user certificates between them.

Note: In this guide, we use sample applications written in Java connecting using client bindings. If you plan to use Java applications using local bindings or C applications, you must create a CMS keystore and certificates using the **`runmqakm`** command. This is shown in the **Quick Start Guide** (Windows or UNIX).

Procedure

1. Create a directory in which to create your keystore, for example `/home/alice/.mqs`. You might wish to create it in the same directory as used by the **Quick Start Guide** (Windows or UNIX) for your platform.

   Note: This directory is referred to as *keystore-dir* in the following steps
2. Create a new keystore and certificate identifying the user `alice` for use in encryption

   Note: The **`keytool`** command is part of the JRE.

   ```
   keytool -genkey -alias Alice_Java_Cert -keyalg RSA -keystore keystore-dir/keystore.jks -storepass passw0rd
   -dname "CN=alice, O=IBM, C=GB" -keypass passw0rd
   ```

   Note:
   - If your *keystore-dir* contains spaces, you must put quotes round the full name of your keystore
   - It is advisable to use a strong password to secure the keystore.
   - For the purpose of this guide, we are using self-signed certificate which can be created without using a Certificate Authority. For production systems, it is advisable not to use self-signed certificates but instead rely on certificates signed by a Certificate Authority.
   - The `alias` parameter specifies the name for the certificate, which interceptors will look up to receive necessary information.
   - The `dname` parameter specifies the details of the **Distinguished Name** (DN), which must be unique for each user.
3. On UNIX, ensure the keystore is readable

   ```
   chmod +r keystore-dir/keystore.jks
   ```
4. Repeat step1-4 for the user `bob`

Results

The two users `alice` and `bob` each now have a self-signed certificate.

4. Creating keystore.conf:
About this task

You must point IBM MQ Advanced Message Security interceptors to the directory where the key databases and certificates are located. This is done via the `keystore.conf` file, which hold that information in the plain text form. Each user must have a separate `keystore.conf` file. This step should be done for both `alice` and `bob`.

Example

For this scenario, the contents of the `keystore.conf` for `alice` are as follows:

```
JKS.keystore = keystore-dir/keystore
JKS.certificate = Alice_Java_Cert
JKS.encrypted = no
JKS.keystore_pass = passw0rd
JKS.key_pass = passw0rd
JKS.provider = IBMJCE
```

For this scenario, the contents of the `keystore.conf` for `bob` are as follows:

```
JKS.keystore = keystore-dir/keystore
JKS.certificate = Bob_Java_Cert
JKS.encrypted = no
JKS.keystore_pass = passw0rd
JKS.key_pass = passw0rd
JKS.provider = IBMJCE
```

Note:
- The path to the keystore file must be provided with no file extension.
- If you already have a `keystore.conf` file because you have followed the instructions in the Quick Start Guide (Windows or UNIX), you can edit the existing file to add these lines.

5. Sharing certificates:
About this task

Share the certificates between the two keystores so that each user can successfully identify the other. This is done by extracting each user's certificate and importing it into the other user's keystore.

Note: The terms *extract* and *export* are used differently by different certificate tools. For example the IBM GSKit Keyman (ikeyman) tool makes a distinction that you *extract* certificates (public keys) and you *export* private keys. This distinction is extremely important for tools that offer both options, since using *export* by mistake would completely compromise your application by passing on its private key. Because the distinction is so important, the IBM MQ documentation strives to use these terms consistently. However, the Java keytool provides a command line option called *exportcert* that extracts only the public key. For these reasons, the following procedure refers to *extracting* certificates by using the *exportcert* option.

Procedure

1. Extract the certificate identifying `alice`.

   ```
   keytool -exportcert -keystore alice-keystore-dir/keystore.jks -storepass passw0rd
   -alias Alice_Java_Cert -file alice-keystore-dir/Alice_Java_Cert.cer
   ```

2. Import the certificate identifying `alice` into the keystore that `bob` will use. When prompted indicate that you will trust this certificate.

```
keytool -importcert -file alice-keystore-dir/Alice_Java_Cert.cer -alias Alice_Java_Cert
-keystore bob-keystore-dir/keystore.jks -storepass passw0rd
```

3. Repeat the steps for `bob`

Results

The two users `alice` and `bob` are now able to successfully identify each other having created and shared self-signed certificates.

What to do next

Verify that a certificate is in the keystore by running the following commands which print out its details:

```
keytool -list -keystore bob-keystore-dir/keystore.jks -storepass passw0rd -alias Alice_Java_Cert
keytool -list -keystore alice-keystore-dir/keystore.jks -storepass passw0rd -alias Bob_Java_Cert
```

6. Defining queue policy:
About this task

With the queue manager created and interceptors prepared to intercept messages and access encryption keys, we can start defining protection policies on `QM_VERIFY_AMS` using the `setmqspl` command. Refer to setmqspl for more information on this command. Each policy name must be the same as the queue name it is to be applied to.

Example

This is an example of a policy defined on the `TEST.Q` queue, signed by the user `alice` using the `SHA1` algorithm, and encrypted using the 256-bit `AES` algorithm for the user `bob`:

```
setmqspl -m QM_VERIFY_AMS -p TEST.Q -s SHA1 -a "CN=alice,O=IBM,C=GB" -e AES256 -r "CN=bob,O=IBM,C=GB"
```

Note: The DNs match exactly those specified in the receptive user's certificate from the key database.

What to do next

To verify the policy you have defined, issue the following command:

```
dspmqspl -m QM_VERIFY_AMS
```

To print the policy details as a set of `setmqspl` commands, the `-export` flag. This allows storing already defined policies:

```
dspmqspl -m QM_VERIFY_AMS -export >restore_my_policies.bat
```

7. Testing the setup:
Before you begin

Ensure the version of Java you are using has the unrestricted JCE policy files installed.

Go to Unrestricted SDK JCE policy files to download the correct policy files if needed.

Note: The version of Java supplied in the IBM MQ installation already has these policy files. It can be found in *MQ_INSTALLATION_PATH*/java/bin.

About this task

By running different programs under different users you can verify if the application has been properly configured. Refer to the **Quick Start Guide** (Windows or UNIX) for your platform, for details about running programs under different users.

Procedure

1. To run these JMS sample applications, use the CLASSPATH setting for your platform as shown in Environment variables used by IBM MQ classes for JMS to ensure the samples directory is included.
2. As the user `alice`, put a message using a sample application, connecting as a client:
   ```
   java JMSProducer -m QM_VERIFY_AMS -d TEST.Q -h localhost -p 1414 -l AMS.SVRCONN
   ```
3. As the user `bob`, get a message using a sample application, connecting as a client:
   ```
   java JMSConsumer -m QM_VERIFY_AMS -d TEST.Q -h localhost -p 1414 -l AMS.SVRCONN
   ```

Results

If the application has been configured properly for both users, the user `alice` 's message is displayed when `bob` runs the getting application.

Protecting remote queues

To fully protect remote queue connections, the same policy must be set on the remote queue and local queue to which messages are transmitted.

When a message is put into a remote queue, IBM MQ Advanced Message Security intercepts the operation and processes the message according to a policy set for the remote queue. For example, for an encryption policy, the message is encrypted before it is passed to the IBM MQ to handle it. After IBM MQ Advanced Message Security has processed the message put into a remote queue, IBM MQ puts it into associated transmission queue and forwards it to the target queue manager and target queue.

When a GET operation is performed on the local queue, IBM MQ Advanced Message Security tries to decode the message according to the policy set on the local queue. For the operation to succeed, the policy used to decrypt the message must be identical to the one used to encrypt it. Any discrepancy will cause the message to be rejected.

If for any reason both policies cannot be set at the same time, a staged roll-out support is provided. The policy can be set on a local queue with toleration flag on, which indicates that a policy associated with a queue can be ignored when an attempt to retrieve a message from the queue involves a message that does not have the security policy set. In this case, GET will try to decrypt the message, but will allow non-encrypted messages to be delivered. This way policies on remote queues can be set after the local queues has been protected (and tested).

Remember: Remove the toleration flag once the IBM MQ Advanced Message Security roll-out has been completed.

Routing protected messages using IBM Integration Bus

IBM MQ Advanced Message Security can protect messages in an infrastructure where IBM Integration Bus, or WebSphere Message Broker Version 8.0.0.1 (or later) is installed. You should understand the nature of both products before applying security in the IBM Integration Bus environment.

About this task

IBM MQ Advanced Message Security provides end-to-end security of the message payload. This means that only the parties specified as the valid senders and recipients of a message are capable of producing or receiving it. This implies that in order to secure messages flowing through IBM Integration Bus, you can either allow IBM Integration Bus to process messages without knowing their content (Scenario 1) or make it an authorized user able to receive and send messages (Scenario 2).

Scenario 1 - Integration Bus cannot see message content:
Before you begin

You should have your IBM Integration Bus connected to an existing queue manager. Replace `QMgrName` with this existing queue manager name in the commands that follow.

About this task

In this scenario, Alice puts a protected message into an input queue `QIN`. Based on the message property `routeTo`, the message is routed either to *bob's* (`QBOB`),[2] (`QCECIL`), or the default (`QDEF`) queue. The routing is possible because IBM MQ Advanced Message Security protects only the message payload and not its headers and properties which remain unprotected and can be read by IBM Integration Bus. IBM MQ Advanced Message Security is used only by *alice*, *bob* and *cecil*. It is not necessary to install or configure it for the IBM Integration Bus.

IBM Integration Bus receives the protected message from the unprotected alias queue in order to avoid any attempt to decrypt the message. If it were to use the protected queue directly, the message would be put onto the DEAD LETTER queue as impossible to decrypt. The message is routed by IBM Integration Bus and arrives on the target queue unchanged. Therefore it is still signed by the original author (both *bob* and *cecil* only accept messages sent by *alice*) and protected as before (only *bob* and *cecil* can read it). IBM Integration Bus puts the routed message to an unprotected alias. The recipients retrieve the message from a protected output queue where IBM MQ AMS will transparently decrypt the message.

Procedure

1. Configure *alice*, *bob* and *cecil* to use IBM MQ Advanced Message Security as described in the **Quick Start Guide** (Windows or UNIX). Ensure the following steps are completed:
   - Creating and authorizing users
   - Creating Key Database and Certificates
   - Creating keystore.conf
2. Provide *alice's* certificate to *bob* and *cecil*, so *alice* can be identified by them when checking digital signatures on messages.

   Do this by extracting the certificate identifying *alice* to an external file, then adding the extracted certificate to *bob's* and *cecil's* keystores. It is important that you use the method described in **Task 5. Sharing Certificates** in the **Quick Start Guide** (Windows or UNIX).
3. Provide *bob* and *cecil's* certificates to *alice*, so *alice* can send messages encrypted for *bob* and *cecil*.

   Do this using the method specified in the previous step.
4. On your queue manager, define local queues called `QIN`, `QBOB`, `QCECIL` and `QDEF`.
   ```
   DEFINE QLOCAL(QIN)
   ```

2. cecil's

5. Setup the security policy for the QIN queue to an eligible configuration. Use the identical setup for the QBOB, QCECIL and QDEF queues.

   ```
   setmqspl -m QMgrName -p QIN -s SHA1 -a "CN=alice,O=IBM,C=GB"
   -e AES256 -r "CN=bob,O=IBM,C=GB" -r "CN=cecil,O=IBM,C=GB"
   ```

   This scenario assumes the security policy where *alice* is the only authorized sender and *bob* and *cecil* are the recipients.
6. Define alias queues AIN, ABOB and ACECIL referencing local queues QIN, QBOB and QCECIL respectively.

   ```
   DEFINE QALIAS(AIN) TARGET(QIN)
   ```
7. Verify that the security configuration for the aliases specified in the previous step is not present; otherwise set its policy to NONE.

   ```
   dspmqspl -m QMgrName -p AIN
   ```
8. In IBM Integration Bus create a message flow to route the messages arriving on the AIN alias queue to the BOB, CECIL, or DEF node depending on the routeTo property of the message. To do so:
   a. Create an MQInput node called IN and assign the AIN alias as its queue name.
   b. Create MQOutput nodes called BOB, CECIL and DEF, and assign alias queues ABOB, ACECIL and ADEF as their respective queue names.
   c. Create a route node and call it TEST.
   d. Connect the IN node to the input terminal of the TEST node.
   e. Create bob, and cecil output terminals for the TEST node.
   f. Connect the bob output terminal to the BOB node.
   g. Connect the cecil output terminal to the CECIL node.
   h. Connect the DEF node to the default output terminal.
   i. Apply the following rules:

      ```
      $Root/MQRFH2/usr/routeTo/text()="bob"
      $Root/MQRFH2/usr/routeTo/text()="cecil"
      ```
9. Deploy the message flow to the IBM Integration Bus runtime component.
10. Running as the user Alice put a message that also contains a message property called routeTo with a value of either bob or cecil. Running the sample application **amqsstm** will allow you to do this.

    ```
    Sample AMQSSTMA start
    target queue is TEST.Q
    Enter property name
    routeTo
    Enter property value
    bob
    Enter property name

    Enter message text
    My Message to Bob
    Sample AMQSSTMA end
    ```
11. Running as user *bob* retrieve the message from the queue QBOB using the sample application **amqsget**.

Results

When *alice* puts a message on the QIN queue, the message is protected. It is retrieved in protected form by the IBM Integration Bus from the AIN alias queue. IBM Integration Bus decides where to route the message reading the routeTo property which is, as all properties, not encrypted. IBM Integration Bus places the message on the appropriate unprotected alias avoiding its further protection. When received by *bob* or *cecil* from the queue, the message is decrypted and the digital signature is verified.

Scenario 2 - Integration Bus can see message content:

About this task

In this scenario, a group of individuals are allowed to send messages to IBM Integration Bus. Another group are authorized to receive messages which are created by IBM Integration Bus. The transmission between the parties and IBM Integration Bus cannot be eavesdropped.

Remember that IBM Integration Bus reads protection policies and certificates only when a queue is opened, so you must reload the execution group after making any updates to protection policies for the changes to take effect.

```
mqsireload execution-group-name
```

If IBM Integration Bus is considered an authorized party allowed to read or sign the message payload, you must configure IBM MQ Advanced Message Security for the user starting the IBM Integration Bus service. Be aware it is not necessarily the same user who puts/gets the messages onto queues nor the user creating and deploying the IBM Integration Bus applications.

Procedure

1. Configure *alice*, *bob*, *cecil* and *dave* and the IBM Integration Bus service user, to use IBM MQ Advanced Message Security as described in the **Quick Start Guide** (Windows or UNIX). Ensure the following steps are completed:
   - Creating and authorizing users
   - Creating Key Database and Certificates
   - Creating keystore.conf
2. Provide *alice*, *bob*, *cecil* and *dave's* certificates to the IBM Integration Bus service user.

   Do this by extracting each of the certificates identifying *alice*, *bob*, *cecil* and *dave* to external files, then adding the extracted certificates to the IBM Integration Bus keystore. It is important that you use the method described in **Task 5. Sharing Certificates** in the **Quick Start Guide** (Windows or UNIX).
3. Provide the IBM Integration Bus service user's certificate to *alice*, *bob*, *cecil* and *dave*.

   Do this using the method specified in the previous step.

   Note: *Alice* and *bob* need the IBM Integration Bus service user's certificate to encrypt the messages correctly. The IBM Integration Bus service user needs *alice's* and *bob's* certificates to verify authors of the messages. The IBM Integration Bus service user needs *cecil's* and *dave's* certificates to encrypt the messages for them. *cecil* and *dave* need the IBM Integration Bus service user's certificate to verify if the message comes from IBM Integration Bus.
4. Define a local queue named `IN` and define the security policy with *alice* and *bob* specified as authors, and the service user for the IBM Integration Bus specified as recipient:
   ```
   setmqspl -m QMgrName -p IN -s MD5 -a "CN=alice,O=IBM,C=GB" -a "CN=bob,O=IBM,C=GB"
   -e AES256 -r "CN=broker,O=IBM,C=GB"
   ```
5. Define a local queue named `OUT`, and define the security policy with the service user for the IBM Integration Bus specified as author, and *cecil* and *dave* specified as recipients:
   ```
   setmqspl -m QMgrName -p OUT -s MD5 -a "CN=broker,O=IBM,C=GB" -e AES256
   -r "CN=cecil,O=IBM,C=GB" -r "CN=dave,O=IBM,C=GB"
   ```
6. In IBM Integration Bus create a message flow with an `MQInput` and `MQOutput` node. Configure the `MQInput` node to use the `IN` queue and the `MQOutput` node to use the `OUT` queue.
7. Deploy the message flow to the IBM Integration Bus runtime component.
8. Running as user *alice* or *bob* put a message on the queue `IN` using the sample application **`amqsput`**.
9. Running as user *cecil* or *dave* retrieve the message from the queue `OUT` using the sample application **`amqsget`**.

Results

Messages sent by *alice* or *bob* to the input queue `IN` are encrypted allowing only IBM Integration Bus to read it. IBM Integration Bus only accepts messages from *alice* and *bob* and rejects any others. The accepted messages are appropriately processed, then signed and encrypted with *cecil's* and *dave's* keys before being put onto the output queue `OUT`. Only *cecil* and *dave* are capable of reading it, messages not signed by IBM Integration Bus are rejected.

Using IBM MQ AMS with IBM MQ Managed File Transfer

This scenario explains how to configure IBM MQ Advanced Message Security to provide message privacy for data being sent through an IBM MQ Managed File Transfer.

Before you begin

Ensure that you have IBM MQ Advanced Message Security component installed on the IBM MQ installation hosting the queues used by IBM MQ Managed File Transfer that you wish to protect.

If your IBM MQ Managed File Transfer agents are connecting in bindings mode, ensure you also have the GSKit component installed on their local installation.

About this task

When transfer of data between two IBM MQ Managed File Transfer agents is interrupted, possibly confidential data might remain unprotected on the underlying IBM MQ queues used to manage the transfer. This scenario explains how to configure and use IBM MQ Advanced Message Security to protect such data on the IBM MQ Managed File Transfer queues.

In this scenario we consider a simple topology comprising one machine with two IBM MQ Managed File Transfer queues. Agents, `AGENT1` and `AGENT2` sharing a single queue manager, `hubQM`, as is described in the scenario Scenario overview. Both agents are connecting in the same way, either in bindings mode or client mode.

1. Creating certificates:
Before you begin

This scenario uses a simple model where a user `ftagent` in a group `FTAGENTS` is used to run the IBM MQ Managed File Transfer agent processes. If you are using your own user and group names, change the commands accordingly.

About this task

IBM MQ Advanced Message Security uses public key cryptography to sign and/or encrypt messages on protected queues.

Note:
- If your IBM MQ Managed File Transfer agents are running in bindings mode, the commands that you use to create a CMS (Cryptographic Message Syntax) keystore are detailed in the **Quick Start Guide** (Windows or UNIX) for your platform.
- If your IBM MQ Managed File Transfer agents are running in client mode, the commands you will need to create a JKS (Java Keystore) are detailed in the "Quick Start Guide for IBM MQ AMS with Java clients" on page 863.

Procedure
1. Create a self-signed certificate to identify the user `ftagent` as detailed in the appropriate Quick Start Guide. Use a Distinguished Name (DN) as follows:

```
CN=ftagent, OU=MFT, O=IBM, L=Hursley, ST=Hampshire, C=GB
```

2. Create a `keystore.conf` file to identify the location of the keystore and the certificate within it as detailed in the appropriate Quick Start Guide.

2. Configuring message protection:
About this task

You should define a security policy for the data queue used by `AGENT2`, using the **`setmqspl`** command. In this scenario the same user is used to start both agents, and therefore the signer and receiver DN are the same and match the certificate we generated.

Procedure

1. Shut down the IBM MQ Managed File Transfer agents in preparation for protection using the **`fteStopAgent`** command.
2. Create a security policy to protect the `SYSTEM.FTE.DATA.AGENT2` queue.
   ```
   setmqspl -m hubQM -p SYSTEM.FTE.DATA.AGENT2 -s SHA1 -a "CN=ftagent, OU=MFT, O=IBM, L=Hursley, ST=Hampshire, C=GB"
   -e AES128 -r "CN=ftagent, OU=MFT, O=IBM, L=Hursley, ST=Hampshire, C=GB"
   ```
3. Ensure the user running the IBM MQ Managed File Transfer agent process has access to browse the system policy queue and put messages on the error queue.
   ```
   setmqaut -m hubQM -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p ftagent +browse
   setmqaut -m hubQM -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p ftagent +put
   ```
4. Restart your IBM MQ Managed File Transfer agents using the **`fteStartAgent`** command.
5. Confirm that your agents restarted successfully by using the **`fteListAgents`** command and verifying that the agents are in `READY` status.

Results

You are now able to submit transfers from `AGENT1` to `AGENT2`, and the file contents will be transmitted securely between the two agents.

Installing IBM MQ Advanced Message Security

Install IBM MQ Advanced Message Security component on various platforms.

About this task

For complete installation procedures, see Installing IBM MQ Advanced Message Security .

Related tasks:
Uninstalling IBM MQ Advanced Message Security

Auditing on z/OS

IBM MQ Advanced Message Security for z/OS provides a means for optional auditing of MQI operations on policy-protected queues. When enabled, IBM System Management Facility (SMF) audit records are generated for the success and failure of these operations on policy-protected queues. Operations audited include MQPUT, MQPUT1, and MQGET.

Auditing is disabled by default, however, you can activate auditing by configuring _AMS_SMF_TYPE and _AMS_SMF_AUDIT in the configured Language Environment® _CEE_ENVFILE file for the AMS address space. For more information, see Task 24: Create procedures for Advanced Message Security. The _AMS_SMF_TYPE variable is used to designate the SMF record type and is a number between 128 and 255. A SMF record type of 180 is usual, however is not mandatory. Auditing is disabled by specifying a value of 0. The _AMS_SMF_AUDIT variable configures whether audit records are created for MQI operations that are successful, MQI operations that fail, or both. The auditing options can also be dynamically changed while AMS is active using operator commands. For more information, see Operating IBM MQ Advanced Message Security.

The SMF record is defined using subtypes, with subtype 1 being a general auditing event. The SMF record contains all data relevant to the request being processed.

The SMF record is mapped by the CSQ0KSMF macro (note the zero in the macro name), which is provided in the target library SCSQMACS. If you are writing data-reduction programs for SMF data, you can include this mapping macro to aid in the development and customization of SMF post-processing routines.

In the SMF records produced by IBM MQ Advanced Message Security for z/OS, the data is organized into sections. The record consists of:

- a standard SMF header
- a header extension defined by IBM MQ Advanced Message Security for z/OS
- a product section
- a data section

The product section of the SMF record is always present in the records produced by IBM MQ Advanced Message Security for z/OS. The data section varies based on subtype. Currently, one subtype is defined and therefore a single data section is used.

SMF is described in the z/OS System Management Facilities manual (SA22-7630). Valid record types are described in the SMFPRMxx member of your system PARMLIB data set. See SMF documentation for more information.

IBM MQ Advanced Message Security audit report generator (CSQ0USMF)

IBM MQ Advanced Message Security (AMS) for z/OS provides an audit report generator tool called CSQ0USMF which is provided in the installation SCSQAUTH library. Sample JCL to run the CSQ0USMF utility called CSQ40RSM is provided in the installation library SCSQPROC.

As an example, the following JCL dumps SMF type 180 records from an SMF data set, and transfers them to a target data set.

```
//IFAUDUMP EXEC PGM=IFASMFDP
//INDD1 DD DSN=SYSn.MANn.syst,DISP=SHR
//OUTDD1 DD DSN=your.target.dataset,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
INDD(INDD1,OPTIONS(DUMP))
OUTDD(OUTDD1,TYPE(180))
/*
```

You must verify the actual SMF data set names used by your installation. The target data set for the dumped records must have a record format of VBS, and a record length of 32760.

The target data set can then be used as input to the CSQ0USMF utility to produce an AMS audit report. For example:

```
//STEP1 EXEC PGM=CSQ0USMF,
// PARM=('/ -SMFTYPE 180 -M qmgr')
//STEPLIB DD DSN=thlqual.SCSQANLE,DISP=SHR
//    DD DSN=thlqual.SCSQAUTH,DISP=SHR
//SMFIN DD DSN=your.target.dataset,DISP=SHR
//
```

The CSQ0USMF program accepts two optional parameters, which are listed in the following table:

Table 94. CSQ0USMF optional parameters

Parameter	Value	Description
SMFTYPE	nnn	The SMF record type applicable to the audit report. The CSQ0USMF program uses only SMF records that match the SMFTYPE value when generating the report. If you do not specify SMFTYPE, a default value of 180 is used.
M	qmgr	The WMQ queue manager name applicable to the audit report. If you do not specify the -M parameter, the audit report will include all audit records for all queue managers represented in the SMFIN data set.

Using keystores and certificates

To provide transparent cryptographic protection to IBM MQ applications, IBM MQ Advanced Message Security uses the keystore file, where public key certificates and a private key are stored. On z/OS, a SAF key ring is used instead of a keystore file.

In IBM MQ Advanced Message Security, users and applications are represented by public key infrastructure (PKI) identities. This type of identity is used to sign and encrypt messages. The PKI identity is represented by the subject's **distinguished name (DN)** field in a certificate that is associated with signed and encrypted messages. For a user or application to encrypt their messages they require access to the keystore file where certificates and associated private and public keys are stored.

On Windows and UNIX the location of the keystore is provided in the keystore configuration file, which is `keystore.conf` by default. Each IBM MQ Advanced Message Security user must have the keystore configuration file that points to a keystore file. IBM MQ Advanced Message Security accepts the following format of keystore files: `.kdb`, `.jceks`, `.jks`.

The default location of the `keystore.conf` file is:
- On UNIX platforms and IBM i: `$HOME/.mqs/`
- On Windows platforms: `%HOMEDRIVE%%HOMEPATH%\.mqs\keystore.conf`

If you are using a specified keystore filename and location, you should use the following commands
- For Java: `java -D MQS_KEYSTORE_CONF = <path>/<filename> <app_name>`
- For C Client and Server:
  - On UNIX platforms: `export MQS_KEYSTORE_CONF= <path> /<filename>`
  - On Windows platforms: `set MQS_KEYSTORE_CONF= <path> \<filename>`

Related concepts:

"Sender distinguished names" on page 898
The sender distinguished names (DNs) identify users who are authorized to place messages on a queue.

"Recipient distinguished names" on page 899
The recipient distinguished names (DN) identify users who are authorized to retrieve messages from a queue.

Structure of the configuration file

The configuration file points IBM MQ Advanced Message Security to the location of the appropriate keystore.

Each of the five configuration file types has a prefix:

CMS Certificate Management System, configuration entries are prefixed with: `cms.`

PKCS#11
Public Key Cryptography Standard #11, configuration entries are prefixed with: `pkcs11.`

PEM Privacy Enhanced Mail format, configuration entries are prefixed with: `pem.`

JKS Java KeyStore, configuration entries are prefixed with: `jks.`

JCEKS
Java Cryptographic Encryption KeyStore, configuration entries are prefixed with: `jceks.`

Example structures for keystores:

CMS

```
cms.keystore = /<dir>/<keystore_file>
cms.certificate = certificate_label
```

PKCS#11

```
pkcs11.library = <dir>\cryptoki.dll
pkcs11.certificate = <certificatelabel>
pkcs11.token = <tokenlabel>
pkcs11.token_pin = <tokenpin>
pkcs11.secondary_keystore = <dir>\signers
```

PEM

```
pem.private = /<dir>/< keystore_file_private_key >
pem.public = /<dir>/< keystore_file_public_keys >
pem.password = <password>
```

Java JKS

```
jks.keystore = <dir>/Keystore
jks.certificate = <certificate_label >
jks.encrypted = no
jks.keystore_pass = <password>
jks.key_pass = <password>
jks.provider = IBMJCE
```

Java JCEKS

```
jceks.keystore = <dir>/Keystore
jceks.certificate = <certificate_label>
jceks.encrypted = no
jceks.keystore_pass = <password>
jceks.key_pass = <password>
jceks.provider = IBMJCE
```

Table 95. Summary of parameters needed for each configuration file type

Parameters	Configuration file type			
	Java (JKS and JCEKS)	**PEM**	**PKCS#11**	**CMS**
`keystore`	✓			
`private`		✓		
`public`		✓		
`password`		✓		
`library`			✓	
`certificate`	✓		✓	✓
`token`			✓	
`token_pin`			✓	
`secondary_keystore`			✓	
`encrypted`	✓			
`keystore_pass`	✓			
`provider`	✓			

Configuration file parameters are defined as follows:

`keystore`
CMS and Java configuration only. Path to the keystore file.

Important:
- The path to the keystore file must not include the file extension.

`private`
PEM configuration only. File name of a file that contains private key and certificate in PEM format.

`public`
PEM configuration only. File name of a file that contains trusted public certificates in PEM format.

`password`
PEM configuration only. Password that is used to decrypt an encrypted private key.

`library`
PKCS#11 only. Path name of the PKCS#11 library.

`certificate`
CMS, PKCS#11 and Java configuration only. Certificate label.

`token`
PKCS#11 only. Token label.

token_pin
PKCS#11 only. PIN to unlock the token.

secondary_keystore
PKCS#11 only. Path name of the CMS keystore, provided without the `.kdb` extension, that contains anchor certificates (root certificates) required by certificates stored on the PKCS #11 token. The secondary keystore can also contain certificates that are intermediate in the trust chain, as well as recipient certificates that are defined in the privacy security policy. This CMS keystore must be accompanied by a stash file which must be located in the same directory as the secondary keystore.

encrypted
Java configuration only. Status of the password.

keystore_pass
Java configuration only. Password for the keystore file.

Note:
- For the CMS keystore, IBM MQ AMS relies on the stash files (`.sth`), whereas JKS and JCEKS might require a password for both the certificate and the user's private key.
-

  Important: Storing passwords in plain text form is a security risk.

key_pass
Java configuration only. Password for the user's private key.

Important: Storing passwords in plain text form is a security risk.

provider
Java configuration only. The Java security provider that implements cryptographic algorithms required by the keystore certificate.

Note: Currently, IBMJCE is the only provider that is supported by IBM MQ Advanced Message Security.

Important: Information that is stored in the keystore is crucial for the secure flow of data that is sent by using IBM MQ. Security administrators must pay particular attention when they are assigning file permissions to these files.

For more information about managing the keystore by using GSKit commands, see GSKCapiCmd program users guide.

Example of the `keystore.conf` file:

```
cms.keystore = c:\Documents and Settings\Alice\AliceKeystore
cms.certificate = AliceCert

jceks.keystore = c:/Documents and Settings/Alice/AliceKeystore
jceks.certificate = AliceCert
jceks.encrypted = no
jceks.keystore_pass = <passw0rd >
jceks.key_pass = <passw0rd >
jceks.provider = IBMJCE
```

Related tasks:

"Protecting passwords in Java" on page 889
Storing keystore and private key passwords as plain text poses a security risk so IBM MQ Advanced Message Security provides a tool that can scramble those passwords using a user's key, which is available in the keystore file.

Message Channel Agent (MCA) interception

MCA interception enables a queue manager running under IBM MQ to selectively enable policies to be applied for server connection channels.

MCA interception allows clients that remain outside IBM MQ Advanced Message Security to still be connected to a queue manager and their messages to be encrypted and decrypted.

MCA interception is intended to provide IBM MQ AMS capability when IBM MQ AMS cannot be enabled at the client. Note that using MCA interception and an IBM MQ AMS-enabled client leads to double-protection of messages which might be problematic for receiving applications. For more information, see "Environment variables used to disable IBM MQ AMS at the client" on page 880.

Keystore configuration file

By default, the keystore configuration file for MCA interception is `keystore.conf` and is located in the `.mqs` directory in the HOME directory path of the user who started the queue manager or the listener. The keystore can also be configured by using the MQS_KEYSTORE_CONF environment variable. For more information about configuring the IBM MQ AMS keystore, see "Using keystores and certificates" on page 874.

To enable MCA interception, you must provide the name of a channel that you want to use in the keystore configuration file. In the specific case of MCA Interception, only a cms keystore type can be used.

For an example on setting up MCA interception, see "IBM MQ Advanced Message Security MCA interception example" on page 879.

Attention: You must complete client authentication and encryption on the selected channels, for example, by using SSL and SSLPEER or CHLAUTH TYPE(SSLPEERMAP), to ensure that only authorized clients can connect and use this capability.

IBM i

If your enterprise uses IBM i, and you selected a commercial Certificate Authority (CA) to sign your certificate, the Digital Certificate Manager creates a certificate request in PEM (Privacy-Enhanced Mail) format. You must forward the request to your chosen CA.

To do this, you must use the following command to select the correct certificate for the channel specified in `channelname`:

```
pem.certificate.channel.<channelname>
```

IBM MQ Advanced Message Security MCA interception example

An example task on how you setup an IBM MQ AMS MCA interception.

Before you begin

Attention: You must complete client authentication and encryption on the selected channels, for example, by using SSL and SSLPEER or CHLAUTH TYPE(SSLPEERMAP), to ensure that only authorized clients can connect and use this capability.

IBM i If your enterprise uses IBM i, and you selected a commercial Certificate Authority (CA) to sign your certificate, the Digital Certificate Manager creates a certificate request in PEM (Privacy-Enhanced Mail) format. You must forward the request to your chosen CA.

About this task

This task takes you through the process of setting up your system to use MCA interception, then verifying the setup.

Note: Prior to IBM WebSphere MQ Version 7.5, IBM MQ AMS was an add-on product that needed to be separately installed and interceptors configured to protect applications. From Version 7.5 onwards, the interceptors are automatically included and dynamically enabled in the MQ client and server runtime environments. In this MCA interception example, the interceptors are provided at the server end of the channel, and an older client runtime is used (in Step 12) to put an unprotected messages across the channel so that it can be seen to be protected by the MCA interceptors. If this example had used a Version 7.5 or later client, it would cause the message to be protected twice, because the MQ client runtime interceptor and the MCA interceptor would both protect the message as it comes into MQ.

Attention: Replace `userID` in the code with your user ID.

Procedure

1. Create the key database and certificates by using the following commands to create a shell script. Also, change the **`INSTLOC`** and **`KEYSTORELOC`** or run the required commands. Note that you might not need to create the certificate for `bob`.

```
INSTLOC=/opt/mq80
KEYSTORELOC=/home/testusr/ssl/ams1
mkdir -p $KEYSTORELOC
chmod -R 777 $KEYSTORELOC
chown -R mqm:mqm $KEYSTORELOC
export PATH=$PATH:$INSTLOC/gskit8/bin
echo "PATH = $PATH"
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$INSTLOC/gskit8/lib64

gsk8capicmd_64 -keydb -create -db $KEYSTORELOC/alicekey.kdb -pw passw0rd -stash
gsk8capicmd_64 -keydb -create -db $KEYSTORELOC/bobkey.kdb -pw passw0rd -stash
gsk8capicmd_64 -cert -create -db $KEYSTORELOC/alicekey.kdb -pw passw0rd
-label alice_cert -dn "cn=alice,O=IBM,c=IN" -default_cert yes
gsk8capicmd_64 -cert -create -db $KEYSTORELOC/bobkey.kdb -pw passw0rd
-label bob_cert  -dn "cn=bob,O=IBM,c=IN"  -default_cert yes
```

2. Share the certificates between the two key databases so that each user can successfully identify the other.

   It is important that you use the method described in **Task 5. Sharing Certificates** in the **Quick Start Guide** (Windows or UNIX).

3. Create `keystore.conf` with the following configuration: `Keystore.conf location: /home/userID/ssl/ams1/`

```
cms.keystore = /home/userID/ssl/ams1/alicekey
cms.certificate.channel.SYSTEM.DEF.SVRCONN = alice_cert
```

4. Create and start queue manager `AMSQMGR1`
5. Define a listener with *port* 14567 and *control* `QMGR`

6. Disable channel authority or set the rules for channel authority. See SET CHLAUTH for more information.
7. Stop the queue manager.
8. Set the keystore:

```
export MQS_KEYSTORE_CONF=/home/userID/ssl/ams1/keystore.conf
```

9. Start the queue manager on the same shell.
10. Set the security policy and verify:

```
setmqspl -m AMSQMGR1 -s SHA256 -e AES256 -p TESTQ -a "CN=alice,O=IBM,C=IN"
-r "CN=alice,O=IBM,C=IN"
dspmqspl -m AMSQMGR1
```

    See setmqspl and dspmqspl for more information.
11. Set the channel configuration:

```
export MQSERVER='SYSTEM.DEF.SVRCONN/TCP/127.0.0.1(14567)'
```

12. Run **amqsputc** from an MQ client that does not automatically enable an MCA interceptor; for example an IBM WebSphere MQ Version 7.1 or earlier client. Put the following two messages:

```
/opt/mqm/samp/bin/amqsputc TESTQ TESTQMGR
```

13. Remove the security policy and verify the result:

```
setmqspl -m AMSQMGR1 -p TESTQ -remove
dspmqspl -m AMSQMGR1
```

14. Browse the queue from your IBM MQ Version 8.0 installation:

```
/opt/mq80/samp/bin/amqsbcg TESTQ AMSQMGR1
```

    The browse output shows the messages in encrypted format.
15. Set the security policy and verify the result:

```
setmqspl -m AMSQMGR1 -s SHA256 -e AES256 -p TESTQ -a "CN=alice,O=IBM,C=IN"
-r "CN=alice,O=IBM,C=IN"
dspmqspl -m AMSQMGR1
```

16. Run **amqsgetc** from your IBM MQ Version 8.0 installation:

```
/opt/mqm/samp/bin/amqsgetc TESTQ TESTQMGR
```

Related tasks:

"Quick Start Guide for IBM MQ AMS with Java clients" on page 863
Use this guide to quickly configure IBM MQ Advanced Message Security to provide message security for Java applications connecting using client bindings. By the time you complete it, you will have created a keystore to verify user identities, and defined signing/encryption policies for your queue manager.

Related reference:

"Known limitations" on page 852
Learn about limitations of IBM MQ Advanced Message Security.

Environment variables used to disable IBM MQ AMS at the client

Disable IBM MQ Advanced Message Security (AMS) at the client to prevent interception errors at the queue manager on the server that the client is connecting to. You can disable IBM MQ AMS by setting an environment variable or by setting a security property stanza in the `mqclient.ini` file.

You are using IBM WebSphere MQ Version 7.5 client, or later, and you are attempting to connect to a queue manager on a server from an earlier version of the product, for example, IBM WebSphere MQ Version 7.1. IBM WebSphere MQ Advanced Message Security is automatically enabled in a IBM WebSphere MQ client from Version 7.5, and so, by default the client tries to check the security policies for objects at the queue manager. However, servers on earlier versions of the product do not have IBM WebSphere MQ Advanced Message Security enabled and this causes 2085 (MQRC_UNKNOWN_OBJECT_NAME) error to be reported.

AMQ_DISABLE_CLIENT_AMS environment variable

You need to set this variable in the following cases:
- If you are using Java Runtime Environment (JRE) other than the IBM Java Runtime Environment (JRE)
- If you are using IBM WebSphere MQ Version 7.5, or later, IBM MQ classes for Java or JMS client.

To set the AMQ_DISABLE_CLIENT_AMS environment variable, you must be running IBM WebSphere MQ Version 7.5.0, Fix Pack 4 or later.

Create and set the environment variable AMQ_DISABLE_CLIENT_AMS to TRUE in the environment where the IBM WebSphere MQ Version 7.5, or later, client is running, to disable IBM WebSphere MQ Advanced Message Security at the client side.

For more information on problems with opening IBM MQ AMS protected queues, see "Problems opening protected queues when using JMS" on page 917.

MQS_DISABLE_ALL_INTERCEPT environment variable

You need to set this variable if:
- You are using IBM MQ Version 8.0 with native clients.
- You need to disable IBM MQ AMS at the client.

Create and set the environment variable MQS_DISABLE_ALL_INTERCEPT to TRUE in the environment where the client is running, to disable IBM MQ AMS at the client side.

DisableClientAMS property in the `mqclient.ini` file

For clients from IBM MQ Version 8.0, you can also disable or enable IBM MQ AMS at the client, in the `mqclient.ini` file, by using the property name `DisableClientAMS`, under the **`Security`** stanza.
- To disable IBM MQ AMS:
  ```
  Security:
  DisableClientAMS=Yes
  ```
- To enable IBM MQ AMS:
  ```
  Security:
  DisableClientAMS=No
  ```

Key usage extensions

Key usage extensions place additional restrictions on the way a certificate can be used.

In IBM MQ Advanced Message Security, the key usage must be set as following: for certificates in X.509 V3 or later standard that are used for the quality of protection integrity, if the key usage extensions are set, they must include at least one of the two:
- **`nonRepudiation`**
- **`digitalSignature`**

For the quality of protection privacy, if the key usage extensions are set, they must also include the **`keyEncipherment`** extension.

Related concepts:

"Quality of protection" on page 901
IBM MQ Advanced Message Security data-protection policies imply a quality of protection (QOP).

Certificate validation methods in IBM MQ AMS

You can use IBM MQ Advanced Message Security to detect and reject revoked certificates so that messages on your queues are not protected using certificates that do not fulfill security standards.

IBM MQ AMS allows you to verify a certificate validity by using either Online Certificate Status Protocol (OCSP) or certificate revocation list (CRL).

IBM MQ AMS can be configured for either OCSP or CRL checking or both. If both methods are enabled, then, for performance reasons, IBM MQ AMS uses OCSP for revocation status first. If the revocation status of a certificate is undetermined after the OCSP checking, IBM MQ AMS uses the CRL checking.

Note that both OCSP and CRL checking are enabled by default.

Related concepts:

"Online Certificate Status Protocol (OCSP)"
Online Certificate Status Protocol (OCSP) determines whether a certificate has been revoked and, therefore, helps to determine whether the certificate can be trusted. OCSP is enabled by default.

"Certificate revocation lists (CRLs)" on page 884
CRLs holds a list of certificates that have been marked by Certificate Authority (CA) as no longer trusted for a variety of reasons, for example, the private key has been lost or compromised.

Online Certificate Status Protocol (OCSP)

Online Certificate Status Protocol (OCSP) determines whether a certificate has been revoked and, therefore, helps to determine whether the certificate can be trusted. OCSP is enabled by default.

OCSP is not supported on IBM i sytems.

Enabling OCSP checking for native interceptors of IBM MQ AMS:

Online Certificate Status Protocol (OCSP) checking in IBM MQ Advanced Message Security is enabled by default, based on information in the certificates being used.

Procedure

Add the following options to the keystore configuration file:

Note: All the OCSP stanza are optional and can be specified independently.

Option	**Description**
`ocsp.enable=off`	Enable the OCSP checking if the certificate being checked has an Authority Info Access (AIA) Extension with an PKIX_AD_OCSP access method containing a URI of where the OCSP Responder is located. Possible values: `on` or `off`.
`ocsp.url=<` *`responder_URL`* `>`	The URL address of OCSP responder. If this option is omitted then non-AIA OCSP checking is disabled.
`ocsp.http.proxy.host=<` *`OCSP_proxy`* `>`	The URL address of the OCSP proxy server. If this option is omitted then a proxy is not used for non-AIA online certificate checks.

Option	Description
`ocsp.http.proxy.port=< port_number >`	The OCSP proxy server's port number. If this option is omitted then the default port of 8080 is used.
`ocsp.nonce.generation=on/off`	Generate nonce when querying OCSP. The default value is `off`.
`ocsp.nonce.check=on/off`	Check nonce after receiving a response from OCSP. The default value is `off`.
`ocsp.nonce.size=8`	Nonce size in bytes.
`ocsp.http.get=on/off`	Specify HTTP GET as your request method. If this option is set to `off`, HTTP POST is used. The default value is `off`.
`ocsp.max_response_size=20480`	Maximum size of response from the OCSP responder provided in bytes.
`ocsp.cache_size=100`	Enable internal OCSP response caching and set the limit for the number of cache entries.
`ocsp.timeout=30`	Waiting time for a server response, in seconds, after which IBM MQ Advanced Message Security times-out.
`ocsp.unknown=ACCEPT`	Defines the behavior when an OCSP server cannot be reached within a timeout period. Possible values: • `ACCEPT` Allows the certificate • `WARN` Allows the certificate and logs a warning • `REJECT` Prevents the certificate from being used and logs an error

Enabling OCSP checking in Java:

To enable OCSP checking for Java in IBM MQ Advanced Message Security, modify the `java.security` file or the keystore configuration file.

About this task

There are two ways of enabling OCSP checking in IBM MQ Advanced Message Security:

Using java.security:

Check whether your certificate contains an Authority Information Access (AIA) certificate extension.

Procedure

1. If AIA is not set up or you want to override your certificate, edit the `$JAVA_HOME/lib/security/java.security` file with the following properties:

   ```
   ocsp.responderURL=http://url.to.responder:port
   ocsp.responderCertSubjectName=CN=Example CA,O=IBM,C=US
   ```

   and enable OCSP checking by editing the `$JAVA_HOME/lib/security/java.security` file with the following line:

   ```
   ocsp.enable=true
   ```
2. If AIA is set up, enable OCSP checking by editing the `$JAVA_HOME/lib/security/java.security` file with the following line:

   ```
   ocsp.enable=true
   ```

What to do next

If you are using Java Security Manager, too complete the configuration, add the following Java permission to `lib/security/java.policy`

```
permission java.security.SecurityPermission "getProperty.ocsp.enable";
```

Using keystore.conf:
Procedure

Add the following attribute to the configuration file:

```
ocsp.enable=true
```

Important: Setting this attribute in the configuration file overrides java.security settings.

What to do next

To complete the configuration, add the following Java permissions to `lib/security/java.policy`:

```
permission java.security.SecurityPermission "getProperty.ocsp.enable";
permission java.security.SecurityPermission "setProperty.ocsp.enable";
```

Certificate revocation lists (CRLs)

CRLs holds a list of certificates that have been marked by Certificate Authority (CA) as no longer trusted for a variety of reasons, for example, the private key has been lost or compromised.

To validate certificates, IBM MQ Advanced Message Security constructs a certificate chain that consists of the signer's certificate and the certificate authority's (CA's) certificate chain up to a trust anchor. A trust anchor is a trusted keystore file that contains a trusted certificate or a trusted root certificate that is used to assert the trust of a certificate. IBM MQ AMS verifies the certificate path using a PKIX validation algorithm. When the chain is created and verified, IBM MQ AMS completes the certificate validation which includes validating the issue and expiry date of each certificate in the chain against the current date, checking if the key usage extension is present in the End Entity certificate. If the extension is appended to the certificate, IBM MQ AMS verifies whether **digitalSignature** or **nonRepudiation** are also set. If they are not, the `MQRC_SECURITY_ERROR` is reported and logged. Next, IBM MQ AMS downloads CRLs from files or from LDAP depending on what values were specified in the configuration file. Only CRLs that are encoded in DER format are supported by IBM MQ AMS. If no CRL related configuration is found in the keystore configuration file, IBM MQ AMS performs no CRL validity check. For each CA certificate, IBM MQ AMS queries LDAP for CRLs using Distinguished Names of a CA to find its CRL. The following attributes are included in the LDAP query:

```
certificateRevocationList,
certificateRevocationList;binary,
authorityRevocationList,
authorityRevocationList;binary
deltaRevocationList
deltaRevocationList;binary,
```

Note: `deltaRevocationList` is supported only when it is specified as distribution points.

Enabling certificate validation and certificate revocation list support in native interceptors:

You must modify the keystore configuration file so that IBM MQ Advanced Message Security can download CLRs from the Lightweight Directory Access Protocol (LDAP) server.

About this task

IBM i Enabling certificate validation and certificate revocation list support in native interceptors is not supported for IBM MQ Advanced Message Security on IBM i.

Procedure

Add the following options to the configuration file:

Note: All the CRL stanza are optional and can be specified independently.

Option	Description
`crl.ldap.host=<` *host_name* `>`	LDAP server host name.
`crl.ldap.port=<` *port_number* `>`	LDAP server port number. You can specify up to 11 servers. Multiple LDAP hosts are used to ensure transparent failover in case of LDAP connection failure. It is expected that all LDAP servers are replicas and contain the same data. When the IBM MQ AMS Java interceptor successfully connects to an LDAP server, it does not attempt to download CRLs from the remaining servers provided.
`crl.cdp=off`	Use this option to check or use CRLDistributionPoints extensions in certificates.
`crl.ldap.version=3`	LDAP protocol version number. Possible values: 2 or 3.
`crl.ldap.user=cn=<` *username* `>`	Log in to the LDAP server. If this value is not specified, CRL attributes in LDAP must be world-readable
`crl.ldap.pass=<` *password* `>`	Password for the LDAP server.
`crl.ldap.cache_lifetime=0`	LDAP cache lifetime in seconds. Possible values: 0-86400.
`crl.ldap.cache_size=50`	LDAP cache size. This option can be specified only if the `crl.ldap.cache_lifetime` value is larger than `0`.
`crl.http.proxy.host=some.host.com`	Http proxy server port for CDP CRL retrieval.
`crl.http.proxy.port=8080`	Http proxy server port number.
`crl.http.max_response_size=204800`	The maximum size of CRL, in bytes, that can be retrieved from an HTTP server that is accepted by GSKit.
`crl.http.timeout=30`	Waiting time for a server response, in seconds, after which IBM MQ AMS times outs.
`crl.http.cache_size=0`	HTTP cache size, in bytes.
`crl.unknown=ACCEPT`	Defines the behavior when a CRL server cannot be reached within a timeout period. Possible values: • `ACCEPT` Allows the certificate • `WARN` Allows the certificate and logs a warning • `REJECT` Prevents the certificate from being used and logs an error

Enabling certificate revocation list support in Java:

To enable CRL support in IBM MQ Advanced Message Security, you must modify the keystore configuration file to allow IBM MQ AMS to download CRLs from the Lightweight Directory Access Protocol (LDAP) server and configure the java.security file.

Procedure

1. Add the following options to the configuration file:

Header	Description
`crl.ldap.host=<` *`host_name`* `>`	LDAP host name.
`crl.ldap.port=<` *`port_number`* `>`	LDAP server port number. You can specify up to 11 servers. Multiple LDAP hosts are used to ensure transparent failover in case of LDAP connection failure. It is expected that all LDAP servers are replicas and contain the same data. When the IBM MQ AMS Java interceptor successfully connects to an LDAP server, it does not attempt to download CRLs from the remaining servers provided. Java does not use `crl.ldap.user` and crl.ldaworldp.pass values. It does not use a user and password when connecting to an LDAP server. As a consequence, CRL attributes in LDAP must be world-readable.
`crl.cdp=on/off`	Use this option to check or use CRLDistributionPoints extensions in certificates.

2. Modify the `JRE/lib/security/java.security` file with the following properties:

Property Name	Description
`com.ibm.security.enableCRLDP`	This property takes the following values: `true`, `false`. If it is set to `true`, when doing certificate revocation check, CRLs are located using the URL from CRL distribution points extension of the certificate. If it is set to `false` or not set, checking CRL by using the CRL distribution points extension is disabled.
`ibm.security.certpath.ldap.cache.lifetime`	This property can be used to set the lifetime of entries in the memory cache of LDAP CertStore to a value in seconds. A value of 0 disables the cache; -1 means unlimited lifetime. If not set, the default lifetime is 30 seconds.
`com.ibm.security.enableAIAEXT`	This property takes the following values: `true`, `false`. If it is set to `true`, any Authority Information Access extensions that are found within the certificates of the certificate path being built are examined to determine whether they contain LDAP URIs. For each LDAP URI found, an LDAPCertStore object is created and added to the collection of CertStores that is used to locate other certificates that are required to build the certificate path. If it is set to `false` or not set, additional LDAPCertStore objects are not created.

Enabling certificate revocation lists (CRLs) on z/OS:

IBM MQ Advanced Message Security supports Certificate Revocation List (CRL) checking of the digital certificates used to protect data messages

About this task

When enabled, IBM MQ Advanced Message Security will validate recipient certificates when messages are put to a privacy protected queue, and validate sender certificates when messages are retrieved from a protected queue (integrity or privacy). Validation in this case includes verification that relevant certificates are not registered in a relevant CRL.

IBM MQ Advanced Message Security uses IBM System SSL services to validate sender and recipient certificates. Detailed documentation regarding System SSL certificate validation can be found in the z/OS Cryptographic Services System Secure Sockets Layer Programming manual (SC24-5901).

To enabled CRL checking, you specify the location of a CRL configuration file via the CRLFILE DD in the started task JCL for the AMS address space. A sample CRL configuration file that can be customized is provided in *thlqual*.SCSQPROC(CSQ40CRL). Settings permitted in this file are as follows:

Table 96. IBM MQ Advanced Message Security CRL configuration variables

Variable	Valid values	Description
crl.ldap.host[.n]	*hostname -or- hostname:port*	The ipaddr/hostname of your LDAP server that hosts CRLs of your issuer certificates. If you do not specify a port number for your LDAP server, the port number specified by crl.ldap.port is used.
crl.ldap.port	*port*	The TCP/IP port number of your LDAP server.
crl.ldap.user	*ldap_user*	The LDAP user name to use when connecting to the LDAP server.
crl.ldap.pass	*ldap_password*	The LDAP password associated with the crl.ldap.user.

You can specify multiple LDAP server host names and ports as follows:

```
crl.ldap.host.1 = hostname -or hostname:port
crl.ldap.host.2 = hostname -or hostname:port
crl.ldap.host.3 = hostname -or hostname:port
```

You can specify up to 10 hostnames. If you do not specify a port number for your LDAP servers, the port number specified by crl.ldap.port is used. Each LDAP server must use the same crl.ldap.user/password combination for access.

When the CRLFILE DD is specified the configuration is loaded during initialization of the IBM MQ Advanced Message Security address space and CRL checking is enabled. If the CRLFILE DD is not specified, or the CRL configuration file is unavailable, or invalid, CRL checking is disabled.

IBM MQ AMS performs a CRL check using IBM System SSL certificate validation services as follows:

Table 97. IBM MQ Advanced Message Security CRL checks

Operation	Quality of protection	Certificate(s) checked
PUT	Privacy	Recipient(s)
GET	Integrity/Privacy	Sender

If a message operation fails a CRL check IBM MQ Advanced Message Security performs the following actions:

Table 98. IBM MQ Advanced Message Security CRL check failure behavior

Operation	CRL check failure
PUT	The message is not put to the target queue. A completion code of MQCC_FAILED and a reason code of MQRC_SECURITY_ERROR is returned to the application.
GET	The message is removed from the target queue and moved to the system protection error queue. A completion code of MQCC_FAILED and a reason code of MQRC_SECURITY_ERROR is returned to the application.

IBM MQ AMS for z/OS uses IBM System SSL services to validate certificates, which includes CRL and trust checking. IBM System SSL provides environment variable GSK_CRL_SECURITY_LEVEL to moderate the operation of CRL checking. For example:

```
GSK_CRL_SECURITY_LEVEL=MEDIUM
```

This variable is documented in the z/OS Cryptographic Services System Secure Sockets Layer Programming manual. Valid assignments include:

- LOW - Certificate validation will not fail if the LDAP server cannot be contacted.
- MEDIUM- Certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined.
- HIGH - Certificate validation requires the LDAP server to be contactable and a CRL to be defined.

The IBM System SSL default is MEDIUM. You can set this variable in the configuration file specified via the ENVARS DD in the started task JCL for the AMS address space. A sample environment variable configuration file is provided in *thlqual*.SCSQPROC(CSQ40ENV).

Note: It is the responsibility of administrators to ensure relevant LDAP services are available and to maintain CRL entries for relevant Certificate Authorities.

Protecting passwords in Java

Storing keystore and private key passwords as plain text poses a security risk so IBM MQ Advanced Message Security provides a tool that can scramble those passwords using a user's key, which is available in the keystore file.

Before you begin

The `keystore.conf` file owner must ensure that only the file owner is entitled to read the file. The passwords protection described in this chapter is only an additional measure of protection.

Procedure

1. Edit the `keystore.conf` files to include path to the keystore and users label.

   ```
   jceks.keystore = c:/Documents and Setting/Alice/AliceKeystore
   jceks.certificate = AliceCert
   jceks.provider = IBMJCE
   ```

2. To run the tool, issue:

   ```
   java -cp com.ibm.mq.jmqi.jar com.ibm.mq.ese.config.KeyStoreConfigProtector keystore_password private_key_password
   ```

   An output with encrypted passwords is generated and can be copied to the `keystore.conf` file.

   To copy the output to the `keystore.conf` file automatically, run:

   ```
   java -cp com.ibm.mq.jmqi.jar com.ibm.mq.ese.config.KeyStoreConfigProtector keystore_password private_key_password >>
   ```

   Note:

   For a list of default locations of `keystore.conf` on various platforms, see “Using keystores and certificates” on page 874.

Example

Here is an example of such output:

```
#Fri Jul 30 15:20:29 CEST 2010
jceks.key_pass=MMXh997n5ZOr8uRlJmc5qity9MN2CggGBMKCDxdbn1AyPklvdgTsOLG6X3C1YT7oDzwaqZFlOR4t\r\nmZsc7JGAx8nqqxLnAucdGn0NW
jceks.keystore_pass=OIdeayBnSCfLG4cFuxEVrk6SYyAsdSPpDqgPf16s9s1M04cqZjNbhgjoA2EXonudHZHH+4s2drvQ\r\nCUvQgu9GuaBMJK2F2OjtI
jceks.encrypted=yes
```

Using certificates on z/OS

About this task

IBM MQ Advanced Message Security implements two levels of protection: integrity and privacy. With the integrity level, messages are signed using the private key of the originator (the application doing the MQPUT). Integrity provides detection of message modification, but the message text itself is not encrypted.

With the privacy level, the message is not only signed, but it is also encrypted. The message is encrypted using a symmetric key and an algorithm specified in the relevant IBM MQ Advanced Message Security policy. The symmetric key itself is encrypted with the public key of each recipient (the application doing the MQGET). Public keys are associated with certificates stored in key rings.

When a message that is protected with privacy is dequeued by a recipient application doing an MQGET, the message must be decrypted. Because it was encrypted using the recipient's public key, it must be decrypted using the recipient's private key found in a key ring.

Use of SAF key rings

IBM MQ Advanced Message Security makes use of existing SAF key ring services to define and manage the certificates needed for signing and encryption. Security products that are functionally equivalent to RACF may be used instead of RACF if they provide the same level of support.

Efficient use of key rings can reduce the administration needed to manage the certificates.

After a certificate is generated (or imported), it must be connected to a key ring to become accessible. The same certificate can be connected to more than one key ring.

IBM MQ Advanced Message Security uses two sets of key rings. One set consists of key rings owned by the individual user IDs that originate or receive messages. Each key ring contains the private key associated with the certificate of the owning user ID. The private key of each certificate is used to sign messages for integrity-protected or privacy-protected queues. It is also used to decrypt messages from privacy-protected queues when receiving messages.

The other set is actually a single key ring associated with the AMS address space. For integrity or privacy protection, it contains the chain of signing CA certificates necessary to validate the signature and message-signing certificate of the message originator.

When privacy protection is used, this key ring also contains the certificates of the message recipients. The public keys in these certificates are used to encrypt the symmetric key that was used to encrypt the message data when the message was put to the protected queue. When these messages are retrieved, the private key of relevant recipients is used to decrypt the symmetric key which is then used to decrypt the message data.

IBM MQ Advanced Message Security uses a key ring name of **drq.ams.keyring** when searching for certificates and private keys. This is the case for both the user and the AMS address space key rings.

For an illustration and further explanation of certificates and key ring, and their role in data protection, refer to Summary of the certificate-related operations.

The private key used for signing and decryption can have any label but must be connected as the default certificate.

Digital certificates and key rings are managed in RACF primarily by using the RACDCERT command.

For more information about certificates, labels, and the RACDCERT command, see *z/OS: Security Server RACF Command Language Reference* and *z/OS: Security Server RACF Security Administrator's Guide*.

Authorizing access to the RACDCERT command

Authorization to use the RACDCERT command is a post-installation task that should have been completed by your z/OS system programmer. This task involves granting relevant permissions to the IBM MQ Advanced Message Security security administrator.

As a summary, these commands are needed to allow access to the RACF RACDCERT command:

```
RDEFINE FACILITY IRR.DIGTCERT.* UACC(NONE)
PERMIT IRR.DIGTCERT.* CLASS(FACILITY) ID( admin ) ACCESS(CONTROL)
SETROPTS RACLIST(FACILITY) REFRESH
```

In this example, *admin* specifies the user ID of your security administrator, or any user you want to use the RACDCERT command.

Creating the certificates and key rings

This section documents the steps required to create the certificates and key rings necessary for z/OS users of IBM MQ Advanced Message Security, using a RACF Certificate Authority (CA).

A scenario of a sending application and a receiving application is used to explain the required steps.

In the examples that follow, user1 is the originator of a message and user2 is the recipient. The user ID of the IBM MQ Advanced Message Security address space is WMQAMSD.

All of the commands in the examples shown here are issued from ISPF option 6 by the administrative user ID `admin`.

Defining a local Certificate Authority certificate: If you are using RACF as your CA, you must create a certificate authority certificate, if you have not already done so. The command shown here creates a certificate authority (or signer) certificate. This example creates a certificate called AMSCA to be used when creating subsequent certificates that reflect the identity of IBM MQ Advanced Message Security users and applications.

This command may be modified, specifically `SUBJECTSDN`, to reflect the naming structure and conventions used at your installation:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('AMSCA') O('ibm') C('us'))
KEYUSAGE(CERTSIGN) WITHLABEL('AMSCA')
```

Note: Certificates signed with this local certificate authority certificate show an issuer of CN=AMSCA,O=ibm,C=us when listed with the RACDCERT LIST command.

Creating a digital certificate with a private key: A digital certificate with a private key must be generated for each IBM MQ Advanced Message Security user. In the example shown here, RACDCERT commands are used to generate certificates for user1 and user2, which are signed with the local CA certificate identified by the label AMSCA.

```
RACDCERT ID(user1) GENCERT SUBJECTSDN(CN('user1') O('ibm') C('us'))
WITHLABEL('user1') SIGNWITH(CERTAUTH LABEL('AMSCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)

RACDCERT ID(user2) GENCERT SUBJECTSDN(CN('user2') O('ibm') C('us'))
WITHLABEL('user2') SIGNWITH(CERTAUTH LABEL('AMSCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)

RACDCERT ID(user1) ALTER (LABEL('user1')) TRUST
RACDCERT ID(user2) ALTER (LABEL('user2')) TRUST
```

The RACDCERT ALTER command is required to add the TRUST attribute to the certificate. When a certificate is first created using this procedure, it has a different valid date range than the signing certificate. As a result, RACF marks it as NOTRUST, which means that the certificate is not to be used. Use the RACDCERT ALTER command to set the TRUST attribute.

The KEYUSAGE attributes HANDSHAKE, DATAENCRYPT and DOCSIGN must be specified for certificates used by IBM MQ Advanced Message Security.

Table 99. RACDCERT KEYUSAGE values and indicators

KEYUSAGE Value	**Indicators Set**
HANDSHAKE	digitalSignature and keyEncipherment
DATAENCRYPT	dataEncipherment
DOCSIGN	nonRepudiation
CERTSIGN	keyCertSign and cRLSign

Creating the RACF key rings: The commands shown here create a key ring for RACF-defined user IDs user1, user2, and the IBM MQ Advanced Message Security address space task user WMQAMSD. The key ring name is fixed by IBM MQ Advanced Message Security and must be coded as shown, without quotes. The name is case-sensitive.

```
RACDCERT ID(user1) ADDRING(drq.ams.keyring)
RACDCERT ID(user2) ADDRING(drq.ams.keyring)
RACDCERT ID(WMQAMSD) ADDRING(drq.ams.keyring)
```

Connecting the certificates to the key rings: Connect the user and CA certificates to the key rings:

```
RACDCERT ID(WMQAMSD) CONNECT(CERTAUTH LABEL('AMSCA')
RING(drq.ams.keyring))
RACDCERT ID(user1) CONNECT(ID(user1) LABEL('user1')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
RACDCERT ID(user2) CONNECT(ID(user2) LABEL('user2')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
RACDCERT ID(WMQAMSD) CONNECT(ID(user2) LABEL('user2')
RING(drq.ams.keyring) USAGE(SITE))
```

The certificate containing the private key used for decryption must be connected to the user's key ring as the default certificate.

The RACDCERT USAGE(SITE) attribute prevents the private key from being accessible in the key ring, while the RACDCERT USAGE(PERSONAL) attribute allows the private key to be used, if it exists. User2's certificate must be connected to the AMS address space key ring because its public key is needed to encrypt messages as they are put to the queue. USAGE(SITE) limits exposure of user2's private key.

The CERTAUTH certificate with label AMSCA must be connected to the IBM MQ Advanced Message Security address space key ring because it was used to sign the certificate of user1, who is the message originator. It is used to validate user1's signing certificate.

Key ring verification: The key ring should appear as shown here, after all commands have been entered:

```
RACDCERT ID(user1) LISTRING(drq.ams.keyring)
Digital ring information for user USER1:
Ring:
>drq.ams.keyring<:

Certificate Label Name           Cert Owner   USAGE    DEFAULT
-------------------------------- ------------ -------- -------
user1                            ID(USER1)    PERSONAL YES

RACDCERT ID(user2) LISTRING(drq.ams.keyring)
Digital ring information for user USER2:
Ring:
>drq.ams.keyring<:

Certificate Label Name           Cert Owner   USAGE    DEFAULT
-------------------------------- ------------ -------- -------
user2                            ID(USER2)    PERSONAL YES

RACDCERT ID(WMQAMSD) LISTRING(drq.ams.keyring)
Digital ring information for user WMQAMSD:
Ring:
>drq.ams.keyring<:

Certificate Label Name           Cert Owner   USAGE    DEFAULT
-------------------------------- ------------ -------- -------
AMSCA                            CERTAUTH     CERTAUTH NO
user2                            ID(USER2)    SITE     NO
User1                            ID(USER1)    SITE     NO
```

Listing the individual certificates also shows the ring association.

```
RACDCERT ID(user2) LIST(label('user2'))
Digital certificate information for user USER2:

***
Label: user2
Certificate ID: 2QfH8Pny9/LzpKKFmfFA
Status: TRUST
Start Date: 2010/05/03 22:59:53
End Date:  2011/05/04 22:59:52
Serial Number:
>15<:
Issuer's Name:
>OU=AMSCA.O=ibm.C=us<:
Subject's Name:
>CN=user2.O=ibm.C=us<:
Key Usage: HANDSHAKE, DATAENCRYPT, DOCSIGN
Private Key Type: Non-ICSF
Private Key Size: 1024
Ring Associations:
Ring Owner: USER2
Ring:
>drq.ams.keyring<:
Ring Owner: WMQAMSD
Ring:
>drq.ams.keyring<:
```

To improve performance, the contents of the drq.ams.keyring associated with the AMS address space is cached for the life of the address space. Changes to that key ring do not become effective automatically. The administrator can refresh the cache by either:
- Stopping and restarting the queue manager.
- Using the z/OS MODIFY command:

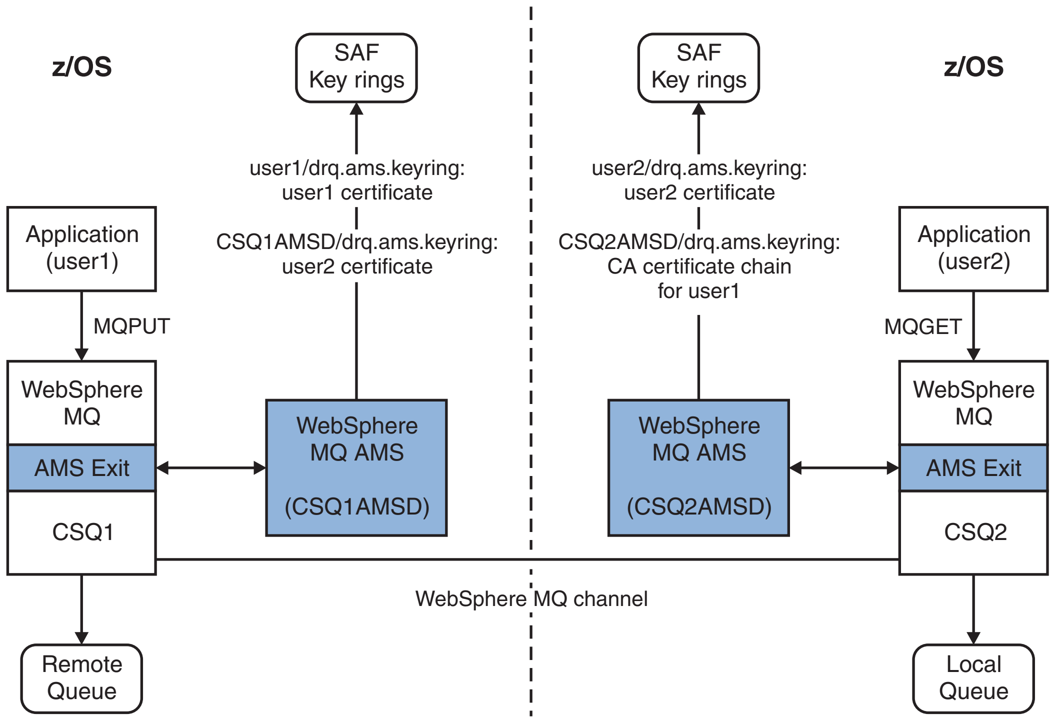
  ```
  F qmgr AMSM,REFRESH KEYRING
  ```

Related information:

Operating IBM MQ Advanced Message Security

Summary of the certificate-related operations

Figure 85 illustrates the relationships between sending and receiving applications and relevant certificates. The scenario illustrated involves remote queuing between two z/OS queue managers using a data-protection policy of privacy. In Figure 85, "AMS" indicates " IBM MQ Advanced Message Security".



z/OS
SAF Key rings
user1/drq.ams.keyring: user1 certificate
CSQ1AMSD/drq.ams.keyring: user2 certificate
Application (user1)
MQPUT
WebSphere MQ
AMS Exit
CSQ1
WebSphere MQ AMS (CSQ1AMSD)
Remote Queue
WebSphere MQ channel
SAF Key rings
z/OS
user2/drq.ams.keyring: user2 certificate
CSQ2AMSD/drq.ams.keyring: CA certificate chain for user1
Application (user2)
MQGET
WebSphere MQ AMS (CSQ2AMSD)
WebSphere MQ
AMS Exit
CSQ2
Local Queue

Figure 85. Application and certificate relationships

In this diagram, an application running as 'user1' puts a message to a remote queue managed by queue manager CSQ1, intended to be retrieved by an application running as 'user2' from a local queue managed by queue manager CSQ2. The diagram assumes an IBM MQ Advanced Message Security policy of privacy, which means the message is both signed and encrypted.

IBM MQ Advanced Message Security intercepts the message when a put occurs and uses user2's certificate (stored in the AMS address space user's key ring) to encrypt a symmetric key used to encrypt the message data.

Note that user2's certificate is connected to the AMS address space user key ring with option USAGE(SITE). This means the AMS address space user can access the certificate and public key, but not the private key.

On the receiving end, IBM MQ Advanced Message Security intercepts the get issued by user2, and uses user2's certificate to decrypt the symmetric key so that it can decrypt the message data. It then validates user1's signature using the CA certificate chain of user1's certificate stored in the AMS address space user's key ring.

Given this scenario, but with a data-protection policy of integrity, certificates for user2 would not be required.

To use IBM MQ Advanced Message Security to enqueue messages on IBM MQ-protected queues having a message protection policy of privacy or integrity, IBM MQ Advanced Message Security must have access to these data items:

- The X.509 V2 or V3 certificate and private key for the user enqueuing the message.
- The chain of certificates used to sign the digital certificates of all message signers.
- If the data protection policy is privacy, the X.509 V2 or V3 certificate of the intended recipients. The intended recipients are listed in the IBM MQ Advanced Message Security policy associated with the queue.

For processes and applications that run on z/OS, IBM MQ Advanced Message Security must have certificates in two places:

- In a SAF-managed key ring associated with the RACF identity of the sending application (the application that enqueues the protected message) or receiving application (if using privacy).

  The certificate that IBM MQ Advanced Message Security locates is the default certificate, and must include the private key. IBM MQ Advanced Message Security assumes the z/OS user identity of the sending application. That is, it acts as a surrogate, so it can access the user's private key.
- In a SAF-managed key ring associated with the AMS address space user.

  When sending messages protected with privacy, this key ring contains the public key certificates of the message recipients. When receiving messages, it contains the chain of Certificate Authority certificates needed to validate the message sender's signature.

The earlier examples shown have used RACF as the local CA. However, you may use another PKI provider (Certificate Authority) at your installation. If you intend to use another PKI product, remember that the private key and the certificate must be imported into a key ring associated with the z/OS RACF user IDs that originate IBM MQ messages protected by IBM MQ Advanced Message Security.

You can use the RACF RACDCERT command as the mechanism to generate certificate requests, which can be exported and sent to the PKI provider of your choice to be issued.

Here is a summary of the certificate-related steps:

1. Request the creation of a CA certificate, one in which RACF is the local CA. Omit this step if you are using another PKI provider.
2. Generate user certificates signed by the CA.
3. Create the key rings for the users and the IBM MQ Advanced Message Security AMS address space ID.
4. Connect the user certificate to the user key ring with the default attribute.
5. Connect the recipients certificates to the IBM MQ Advanced Message Security AMS address space user key ring using the usage(site) attribute (This step is necessary only for user certificates that will ultimately be the recipients of privacy-protected messages).
6. Connect the CA certificate chains for message senders to the IBM MQ Advanced Message Security AMS address space user key ring. (This step is necessary only for AMS tasks that will be verifying sender signatures.)

Configuring a non-z/OS resident PKI

IBM MQ Advanced Message Security for z/OS, uses X.509 V3 digital certificates in the protection-processing of messages placed on or received from IBM MQ queues. IBM MQ Advanced Message Security itself does not create or manage the life cycle of these certificates; that function is provided by a public key infrastructure (PKI). The examples in this publication that illustrate the use of certificates use z/OS Security Server RACF to fill certificate requests.

Whether or not a z/OS or non-z/OS resident PKI is used, IBM MQ AMS for z/OS uses only key rings that are managed by RACF or its equivalent. These key rings are based on Security Authorization Facility (SAF) and are the repository used by IBM MQ AMS for z/OS to retrieve certificates for originators and recipients of messages placed on or received from IBM MQ queues.

For messages that are originated from z/OS, which are protected by either integrity or encryption policy, the certificate and private key of the originating user ID must be stored in an SAF-managed key ring that is associated with the z/OS user ID of the message originator.

RACF includes the capability for importing certificates and private keys into RACF-managed key rings. See the z/OS Security Server RACF publications for the details and examples of how to load certificates to RACF managed key rings.

If your installation is using one of the supported PKI products, refer to the publications that accompany the product for information on how to deploy it.

Security policies

IBM MQ Advanced Message Security uses security policies to specify the cryptographic encryption and signature algorithms for encrypting and authenticating messages that flow through the queues.

Security policies overview

IBM MQ Advanced Message Security security policies are conceptual objects that describe the way a message is cryptographically encrypted and signed.

For details of the security policy attributes, see the following subtopics:

Related concepts:

"Quality of protection" on page 901
IBM MQ Advanced Message Security data-protection policies imply a quality of protection (QOP).

"Security policy attributes" on page 900
You can use IBM MQ Advanced Message Security to select a particular algorithm or method to protect the data.

Policy name

The policy name is a unique name that identifies a specific IBM MQ Advanced Message Security policy and the queue to which it applies.

The policy name must be the same as the queue name to which it applies. There is a one-to-one mapping between an IBM MQ Advanced Message Security (IBM MQ AMS) policy and a queue.

By creating a policy with the same name as a queue, you activate the policy for that queue. Queues without matching policy names are not protected by IBM MQ AMS.

The scope of the policy is relevant to the local queue manager and its queues. Remote queue managers must have their own locally-defined policies for the queues they manage.

Signature algorithm

The signature algorithm indicates the algorithm that should be used when signing data messages.

Valid values include:

- `MD5`
- `SHA-1`
- `SHA-2` family:
  - SHA256
  - SHA384 (minimum key length acceptable - 768 bits)
  - SHA512 (minimum key length acceptable - 768 bits)

A policy that does not specify a signature algorithm, or specifies an algorithm of `NONE`, implies that messages placed on the queue associated with the policy are not signed.

Note: The quality of protection used for the message put and get functions must match. If there is a policy quality of protection mismatch between the queue and the message in the queue, the message is not accepted and is sent to the error handling queue. This rule applies for both local and remote queues.

Encryption algorithm

The encryption algorithm indicates the algorithm that should be used when encrypting data messages placed on the queue associated with the policy.

Valid values include:

- `RC2`
- `DES`
- `3DES`
- `AES128`
- `AES256`

A policy that does not specify an encryption algorithm or specifies an algorithm of `NONE` implies that messages placed on the queue associated with the policy are not encrypted.

Note that a policy that specifies an encryption algorithm other than `NONE` must also specify at least one Recipient DN and a signature algorithm because IBM MQ Advanced Message Security encrypted messages are also signed.

Important: The quality of protection used for the message put and get functions must match. If there is a policy quality of protection mismatch between the queue and the message in the queue, the message is not accepted and is sent to the error handling queue. This rule applies for both local and remote queues.

Toleration

The toleration attribute indicates whether IBM MQ Advanced Message Security can accept messages with no security policy specified.

When retrieving a message from a queue with a policy to encrypt messages, if the message is not encrypted, it is returned to the calling application. Valid values include:

0	No (**default**).
1	Yes.

A policy that does not specify a toleration value or specifies 0, implies that messages placed on the queue associated with the policy must match the policy rules.

Toleration is optional and exists to facilitate configuration roll-out, where policies were applied to queues but those queues already contain messages that do not have a security policy specified.

Sender distinguished names

The sender distinguished names (DNs) identify users who are authorized to place messages on a queue.

IBM MQ Advanced Message Security (IBM MQ AMS) does not check whether a message has been placed on a data-protected queue by a valid user until the message is retrieved. At this time, if the policy stipulates one or more valid senders, and the user that placed the message on the queue is not in the list of valid senders, IBM MQ AMS returns an error to the getting application, and place the message on its error queue.

A policy can have 0 or more sender DNs specified. If no sender DNs are specified for the policy, any user can put data-protected messages to the queue providing the user's certificate is trusted.

Sender distinguished names have the following form:

```
CN=Common Name,O=Organization,C=Country
```

Important:

- All DNs must be in uppercase and in the same order as listed in the table.

Component name	Value
CN	The common name for the object of this DN, such as a full name or the intended purpose of a device.
OU	The unit within the organization with which the object of the DN is affiliated, such as a corporate division or a product name.
O	The organization with which the object of the DN is affiliated, such as a corporation.
L	The locality (city or municipality) where the object of the DN is located.
ST	The state or province name where the object of the DN is located.
C	The country where the object of the distinguished name (DN) is located.

- If one or more sender DNs are specified for the policy, only those users can put messages to the queue associated with the policy.
- Sender DNs, when specified, must match exactly the DN contained in the digital certificate associated with user putting the message.
- IBM MQ AMS supports DNs with values only from Latin-1 character set. To create DNs with characters of the set, you must first create a certificate with a DN that is created in UTF-8 coding using

UNIX platforms with UTF-8 coding turned on or with the iKeyman utility. Then you must create a policy from a UNIX platform with UTF-8 coding turned on or use the IBM MQ AMS plug-in to IBM MQ.

- The method used by IBM MQ AMS, to convert the name of the sender from x.509 format to DN format, always uses ST= for the State or Province value.
- The following special characters need escape characters:

```
    , (comma)
    + (plus)
    " (double quote)
    \ (backslash)
    < (less than)
    > (greater than)
    ; (semicolon)
```

- If the Distinguished Name contains embedded blanks, you should enclose the DN in double quotation marks.

Recipient distinguished names

The recipient distinguished names (DN) identify users who are authorized to retrieve messages from a queue.

A policy can have zero or more recipient DNs specified. Recipient distinguished names have the following form:

```
CN=Common Name,O=Organization,C=Country
```

Important:

- All DNs must be in uppercase and in the same order as listed in the table.

Component name	Value
CN	The common name for the object of this DN, such as a full name or the intended purpose of a device.
OU	The unit within the organization with which the object of the DN is affiliated, such as a corporate division or a product name.
O	The organization with which the object of the DN is affiliated, such as a corporation.
L	The locality (city or municipality) where the object of the DN is located.
ST	The state or province name where the object of the DN is located.
C	The country where the object of the distinguished name (DN) is located.

- If no recipient DNs are specified for the policy, any user can get messages from the queue associated with the policy.
- If one or more recipient DNs are specified for the policy, only those users can get messages from the queue associated with the policy.
- Recipient DNs, when specified, must match exactly the DN contained in the digital certificate associated with user getting the message.
- IBM MQ Advanced Message Security supports DNs with values only from Latin-1 character set. To create DNs with characters of the set, you must first create a certificate with a DN that is created in UTF-8 coding using UNIX platforms with UTF-8 coding turned on or with the iKeyman utility. Then you must create a policy from a UNIX platform with UTF-8 coding turned on or use the IBM MQ Advanced Message Security plug-in to IBM MQ.

Security policy attributes

You can use IBM MQ Advanced Message Security to select a particular algorithm or method to protect the data.

A security policy is a conceptual object that describes the way a message is cryptographically encrypted and signed. The following table presents the security policy attributes in IBM MQ Advanced Message Security:

Attributes	Description
Policy name	Unique name of the policy for a queue manager.
Signature algorithm	Cryptographic algorithm that is used to sign messages before sending.
Encryption algorithm	Cryptographic algorithm that is used to encrypt messages before sending.
Recipient list	List of certificate distinguished names (DNs) of potential receivers of a message.
Signature DN checklist	List of signature DNs to be validated during message retrieval.

In IBM MQ Advanced Message Security, messages are encrypted with a symmetric key, and the symmetric key is encrypted with the public keys of the recipients. Public keys are encrypted with the RSA algorithm, with keys of an effective length up to 2048 bits. The actual asymmetric key encryption depends on the certificate key length.

The supported symmetric-key algorithms are as follows:

- `RC2`
- `DES`
- `3DES`
- `AES128`
- `AES256`

IBM MQ Advanced Message Security also supports the following cryptographic hash functions:

- `MD5`
- `SHA-1`
- `SHA-2` family:
  - SHA256
  - SHA384 (minimum key length acceptable - 768 bits)
  - SHA512 (minimum key length acceptable - 768 bits)

Note: The quality of protection used for the message put and get functions must match. If there is a policy quality of protection mismatch between the queue and the message in the queue, the message is not accepted and is sent to the error handling queue. This rule applies for both local and remote queues.

Quality of protection

IBM MQ Advanced Message Security data-protection policies imply a quality of protection (QOP).

The three quality of protection levels in IBM MQ Advanced Message Security depend on cryptographic algorithms that are used to sign and encrypt the message:

- Privacy - messages placed on the queue must be signed and encrypted.
- Integrity - messages placed on the queue must be signed by the sender.
- None - no data protection is applicable.

A policy that stipulates that messages must be signed when placed on a queue has a QOP of INTEGRITY. A QOP of INTEGRITY means that a policy stipulates a signature algorithm, but does not stipulate an encryption algorithm. Integrity-protected messages are also referred to as "SIGNED".

A policy that stipulates that messages must be signed and encrypted when placed on a queue has a QOP of PRIVACY. A QOP of PRIVACY means that when a policy stipulates a signature algorithm and an encryption algorithm. Privacy-protected messages are also referred to as "SEALED".

A policy that does not stipulate a signature algorithm or an encryption algorithm has a QOP of NONE. IBM MQ Advanced Message Security provides no data-protection for queues that have a policy with a QOP of NONE.

Managing security policies

A security policy is a conceptual object that describes the way a message is cryptographically encrypted and signed.

All administrative tasks related to security policies are run from the following location:

- On UNIX platforms: `<MQInstallRoot>/bin`
- On Windows platforms administrative tasks can be run from any location as the `PATH` environment variable is updated at the installation.

  On UNIX platforms, and Windows, you use the DELETE POLICY, DISPLAY POLICY, and SET POLICY (or equivalent PCF) commands to manage your security policies.
- On IBM i, the DSPMQMSPL, SETMQMSPL, and WRKMQMSPL commands are installed into the QSYS system library for the primary language of the system when IBM MQ is installed.

  Additional national language versions get installed into QSYS29xx libraries according to the language feature load.

  For example, a machine with US English as the primary language and Korean as the secondary language has the US English commands installed into QSYS and the Korean secondary language load in QSYS2962 as 2962 is the language load for Korean.
- On z/OS, the administrative commands are run using the message security policy utility (CSQ0UTIL). When policies are created, modified or deleted on z/OS, the changes are not recognized by IBM MQ Advanced Message Security until the queue manager is stopped and restarted, or the z/OS MODIFY command is used to refresh the IBM MQ Advanced Message Security policy configuration. For example:

  ```
  F qmgr AMSM,REFRESH POLICY
  ```

Related tasks:

"Creating security policies"
Security policies define the way in which a message is protected when the message is put, or how a message must have been protected when a message is received.

"Changing security policies" on page 903
You can use IBM MQ Advanced Message Security to alter details of security policies that you have already defined.

"Displaying and dumping security policies" on page 903
Use the `dspmqspl` command to display a list of all security policies or details of a named policy depending on the command-line parameters you supply.

"Removing security policies" on page 905
To remove security policies in IBM MQ Advanced Message Security, you must use the `setmqspl` command.

Related information:

The message security policy utility (CSQ0UTIL)

Operating IBM MQ Advanced Message Security

Creating security policies

Security policies define the way in which a message is protected when the message is put, or how a message must have been protected when a message is received.

Before you begin

There are some entry conditions which must be met when creating security policies:

- The queue manager must be running.
- The name of a security policy must follow Rules for naming IBM MQ objects.
- You must have the necessary authority to connect to the queue manager and create a security policy. On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL). On other platforms other than z/OS, you must grant the necessary +connect, +inq and +chg authorities using the setmqaut command. For more information about configuring security see "Setting up security" on page 511.
- On z/OS, ensure the required system objects have been defined according to the definitions in CSQ4INSM.

Example

Here is an example of creating a policy on queue manager `QMGR`. The policy specifies that messages be signed using the `SHA1` algorithm and encrypted using the `AES256` algorithm for certificates with DN: CN=joe,O=IBM,C=US and DN: CN=jane,O=IBM,C=US. This policy is attached to `MY.QUEUE`:

```
setmqspl -m QMGR -p MY.QUEUE -s SHA1 -e AES256 -r CN=joe,O=IBM,C=US -r CN=jane,O=IBM,C=US
```

Here is an example of creating policy on the queue manager `QMGR`. The policy specifies that messages be encrypted using the `DES` algorithm for certificates with DNs: CN=john,O=IBM,C=US and CN=jeff,O=IBM,C=US and signed with the `MD5` algorithm for certificate with DN: CN=phil,O=IBM,C=US

```
setmqspl -m QMGR -p MY.OTHER.QUEUE -s MD5 -e DES -r CN=john,O=IBM,C=US -r CN=jeff,O=IBM,C=US -a CN=phil,O=IBM,C=US
```

Note:

- The quality of protection being used for the message put and get must match. If the policy quality of protection that is defined for the message is weaker than that defined for a queue, the message is sent to the error handling queue. This policy is valid for both local and remote queues.

Related reference:

Complete list of setmqspl command attributes

Changing security policies

You can use IBM MQ Advanced Message Security to alter details of security policies that you have already defined.

Before you begin

- The queue manager on which you want to operate must be running.
- You must have the necessary authority to connect to the queue manager and create a security policy. On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL). On other platforms other than z/OS, you must grant the necessary +connect, +inq and +chg authorities using the setmqaut command. For more information about configuring security see "Setting up security" on page 511.

About this task

To change security policies, apply the `setmqspl` command to an already existing policy providing new attributes.

Example

Here is an example of creating a policy named `MYQUEUE` on a queue manager named `QMGR` specifying that messages will be encrypted using the `RC2` algorithm for certificates with DN:CN=bob,O=IBM,C=US and signed with the `SHA1` algorithm for certificates with DN:CN=jeff,O=IBM,C=US.

```
setmqspl -m QMGR -p MYQUEUE -e RC2 -s SHA1 -a CN=jeff,O=IBM,C=US -r CN=alice,O=IBM,C=US
```

To alter this policy, issue the `setmqspl` command with all attributes from the example changing only the values you want to modify. In this example, previously created policy is attached to a new queue and its encryption algorithm is changed to `AES256`:

```
setmqspl -m QMGR -p MYQUEUE -e AES256 -s SHA1 -a CN=jeff,O=IBM,C=US -r CN=alice,O=IBM,C=US
```

Related reference:

Complete list of setmqspl command attributes

Displaying and dumping security policies

Use the `dspmqspl` command to display a list of all security policies or details of a named policy depending on the command-line parameters you supply.

Before you begin

- To display security policies details, the queue manager must exist, and be running.
- You must have the necessary authority to connect to the queue manager and create a security policy. On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL). On other platforms other than z/OS, you must grant the necessary +connect, +inq and +chg authorities using the setmqaut command. For more information about configuring security see "Setting up security" on page 511.

About this task

Here is the list of `dspmqspl` command flags:

Table 100. `dspmqspl` command flags.

Command flag	**Explanation**
`-m`	Queue manager name (**mandatory**).
`-p`	Policy name.
`-export`	Adding this flag generates output which can easily be applied to a different queue manager.

Example

In this example we will create two security policies for `venus.queue.manager`:

```
setmqspl -m venus.queue.manager -p AMS_POL_04_ONE -s MD5 -a "CN=signer1,O=IBM,C=US" -e NONE
setmqspl -m venus.queue.manager -p AMS_POL_06_THREE -s MD5 -a "CN=another signer,O=IBM,C=US" -e NONE
```

This example shows a command that displays details of all policies defined for `venus.queue.manager` and the output it produces:

```
dspmqspl -m  venus.queue.manager

Policy Details:
Policy name: AMS_POL_04_ONE
Quality of protection: INTEGRITY
Signature algorithm: MD5
Encryption algorithm: NONE
Signer DNs:
  CN=signer1,O=IBM,C=US
Recipient DNs: -
Toleration: 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
Policy Details:
Policy name: AMS_POL_06_THREE
Quality of protection: INTEGRITY
Signature algorithm: MD5
Encryption algorithm: NONE
Signer DNs:
  CN=another signer,O=IBM,C=US
Recipient DNs: -
Toleration: 0
```

This example shows a command that displays details of a selected security policy defined for `venus.queue.manager` and the output it produces:

```
dspmqspl -m venus.queue.manager -p AMS_POL_06_THREE

Policy Details:
Policy name: AMS_POL_06_THREE
Quality of protection: INTEGRITY
Signature algorithm: MD5
Encryption algorithm: NONE
Signer DNs:
  CN=another signer,O=IBM,C=US
Recipient DNs: -
Toleration: 0
```

In the next example, first, we create a security policy and then, we export the policy using the `-export` flag:

```
setmqspl -m venus.queue.manager -p AMS_POL_04_ONE -s MD5 -a "CN=signer1,O=IBM,C=US" -e NONE

dspmqspl -m venus.queue.manager -export
```

On z/OS, the exported policy information is written by CSQ0UTIL to the EXPORT DD. On platforms other than z/OS, redirect the output to a file, for example:

```
dspmqspl -m venus.queue.manager -export > policies.[bat|sh]
```

To import a security policy:

- On Windows platforms, run `policies.bat`
- On UNIX platforms:
  1. Log on as a user that belongs to the `mqm` IBM MQ administration group.
  2. Issue `. policies.sh`.
- On z/OS use the CSQ0UTIL utility, specifying to SYSIN the data set containing the exported policy information.

Related reference:

Complete list of dspmqspl command attributes

Removing security policies

To remove security policies in IBM MQ Advanced Message Security, you must use the `setmqspl` command.

Before you begin

There are some entry conditions which must be met when managing security policies:

- The queue manager must be running.
- You must have the necessary authority to connect to the queue manager and create a security policy. On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL). On other platforms other than z/OS, you must grant the necessary +connect, +inq and +chg authorities using the setmqaut command. For more information about configuring security see “Setting up security” on page 511.

About this task

Use the `setmqspl` command with the `-remove` option.

Example

Here is an example of removing a policy:

```
setmqspl -m QMGR -remove -p MY.OTHER.QUEUE
```

Related reference:

Complete list of setmqspl command attributes

System queue protection

System queues enable communication between IBM MQ and its ancillary applications. Whenever a queue manager is created, a system queue is also created to store IBM MQ internal messages and data. You can protect system queues with IBM MQ Advanced Message Security so that only authorized users can access or decrypt them.

System queue protection follows the same pattern as the protection of regular queues. See “Creating security policies” on page 902.

To use system queue protection on Windows platforms, copy the `keystore.conf` file to the following directory:

```
c:\Documents and Settings\Default User\.mqs\keystore.conf
```

On platforms other than z/OS, to provide protection for `SYSTEM.ADMIN.COMMAND.QUEUE`, the command server must have access to the `keystore` and the `keystore.conf`, which contain keys and a configuration so that the command server can access keys and certificates. All changes made to the security policy of `SYSTEM.ADMIN.COMMAND.QUEUE` require the restart of the command server.

All messages that are sent and received from the command queue are signed or signed and encrypted depending on policy settings. If an administrator defines authorized signers, command messages that do not pass the signer Distinguished Name (DN) check are not executed by the command server and are not routed to the IBM MQ Advanced Message Security error handling queue. Messages that are sent as replies to IBM MQ Explorer temporary dynamic queues are not protected by IBM MQ AMS.

Security policies do not have an effect on the following SYSTEM queues:
- SYSTEM.ADMIN.ACCOUNTING.QUEUE
- SYSTEM.ADMIN.ACTIVITY.QUEUE
- SYSTEM.ADMIN.CHANNEL.EVENT
- SYSTEM.ADMIN.COMMAND.EVENT
- z/OS SYSTEM.ADMIN.COMMAND.QUEUE
- SYSTEM.ADMIN.CONFIG.EVENT
- SYSTEM.ADMIN.LOGGER.EVENT
- SYSTEM.ADMIN.PERFM.EVENT
- SYSTEM.ADMIN.PUBSUB.EVENT
- SYSTEM.ADMIN.QMGR.EVENT
- SYSTEM.ADMIN.STATISTICS.QUEUE
- SYSTEM.ADMIN.TRACE.ROUTE.QUEUE
- SYSTEM.AUTH.DATA.QUEUE
- SYSTEM.BROKER.ADMIN.STREAM
- z/OS SYSTEM.BROKER.CLIENTS.DATA
- SYSTEM.BROKER.CONTROL.QUEUE
- SYSTEM.BROKER.DEFAULT.STREAM
- SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS
- z/OS SYSTEM.BROKER.SUBSCRIPTIONS.DATA
- SYSTEM.CHANNEL.INITQ
- SYSTEM.CHANNEL.SYNCQ
- z/OS SYSTEM.CHLAUTH.DATA.QUEUE
- SYSTEM.CICS.INITIATION.QUEUE
- SYSTEM.CLUSTER.COMMAND.QUEUE
- SYSTEM.CLUSTER.HISTORY.QUEUE
- SYSTEM.CLUSTER.REPOSITORY.QUEUE
- SYSTEM.CLUSTER.TRANSMIT.QUEUE
- z/OS SYSTEM.COMMAND.INPUT
- z/OS SYSTEM.DDELAY.LOCAL.QUEUE
- SYSTEM.DEAD.LETTER.QUEUE
- SYSTEM.DURABLE.SUBSCRIBER.QUEUE
- SYSTEM.HIERARCHY.STATE
- SYSTEM.INTER.QMGR.CONTROL
- SYSTEM.INTER.QMGR.FANREQ
- SYSTEM.INTER.QMGR.PUBS

- SYSTEM.INTERNAL.REPLY.QUEUE
- z/OS SYSTEM.JMS.PS.STATUS.QUEUE
- z/OS SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.PENDING.DATA.QUEUE
- SYSTEM.PROTECTION.ERROR.QUEUE
- SYSTEM.PROTECTION.POLICY.QUEUE
- z/OS SYSTEM.QSG.CHANNEL.SYNCQ
- z/OS SYSTEM.QSG.TRANSMIT.QUEUE
- z/OS SYSTEM.QSG.UR.RESOLUTION.QUEUE
- SYSTEM.RETAINED.PUB.QUEUE
- z/OS SYSTEM.RETAINED.PUB.QUEUE
- SYSTEM.SELECTION.EVALUATION.QUEUE
- SYSTEM.SELECTION.VALIDATION.QUEUE

Granting OAM permissions

File permissions authorize all users to execute `setmqspl` and `dspmqspl` commands. However, IBM MQ Advanced Message Security relies on the Object Authority Manager (OAM) and every attempt to execute these commands by a user who does not belong to the mqm group, which is the IBM MQ administration group, or does not have permissions to read security policy settings that are granted, results in an error.

Procedure

To grant necessary permissions to a user, run:

```
setmqaut -m SOME.QUEUE.MANAGER -t qmgr -p SOME.USER +connect +inq
setmqaut -m SOME.QUEUE.MANAGER -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p SOME.USER +browse +put
setmqaut -m SOME.QUEUE.MANAGER -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p SOME.USER +put
```

Note: You only need to set these OAM authorities if you intend to connect clients, to the queue manager, using IBM MQ Advanced Message Security Version 7.0.1.

Granting security permissions

When using command resource security you must set up permissions to allow IBM MQ Advanced Message Security to function. This topic uses RACF commands in the examples. If your enterprise uses a different external security manager (ESM) you must use the equivalent commands for that ESM.

There are three aspects to granting security permissions:
- "The AMSM address space" on page 908
- "CSQ0UTIL" on page 908
- "Using queues that have an IBM MQ Advanced Message Security policy defined." on page 908

Notes: The example commands use the following variables.
1. *QMgrName* - the name of the queue manager. On z/OS, this value can also be the name of a queue-sharing group.
2. *username* - this can be a group name.
3. The examples show the MQQUEUE class. this can also be MXQUEUE, GMQQUEUE or GMXQUEUE. See "Profiles for queue security" on page 581 for further information.

Furthermore, if the profile already exists, you do not require the RDEFINE command.

The AMSM address space

You need to issue some IBM MQ security to the user name that the IBM MQ Advanced Message Security address space runs under.

- For batch connection to the queue manager, issue

```
RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
            PERMIT QMgrName.BATCH CLASS(MQCONN) ID(username) ACCESS(READ)
```

- For access to the SYSTEM.PROTECTION.POLICY.QUEUE, issue:

```
RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
            PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

CSQ0UTIL

The utility that allows users to run the **setmqspl** and **dspmqspl** commands requires the following permissions, where the user name is the job user ID:

- For batch connection to the queue manager, issue:

```
  RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
             PERMIT QMgrName.BATCH CLASS(MQCONN) ID(username) ACCESS(READ)
```

- For access to the SYSTEM.PROTECTION.POLICY.QUEUE, required for the **setmqpol** command, issue:

```
  RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
              PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(ALTER)
```

- For access to the SYSTEM.PROTECTION.POLICY.QUEUE, required for the **dspmqpol** command, issue:

```
  RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
                PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

Using queues that have an IBM MQ Advanced Message Security policy defined.

When an application does any work with queues that have a policy defined on them, that application requires additional permissions to allow IBM MQ Advanced Message Security to protect messages.

The application requires:

- Read access to the SYSTEM.PROTECTION.POLICY.QUEUE. Do this by issuing:

```
RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
          PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

- Put access to the SYSTEM.PROTECTION.ERROR.QUEUE. Do this by issuing:

```
RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.ERROR.QUEUE UACC(NONE)
            PERMIT QMgrName.SYSTEM.PROTECTION.ERROR.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

IBM i: Setting up certificates and the keystore configuration file

Your first task when setting up IBM MQ Advanced Message Security protection is to create a certificate, and associate that with your environment. The association is configured through a file held in the integrated filesystem (IFS).

Procedure

1. To create a self-signed certificate using the OpenSSL tooling shipped with IBM i, issue the following command from QShell:

   ```
   /QOpenSys/usr/bin/openssl req -x509 -newkey rsa:2048 -keyout
   $HOME/private.pem -out $HOME/mycert.pem -nodes -days 365
   ```

   The command prompts for various distinguished name attributes for a new self-signed certificate, including:
   - Common Name (CN=)
   - Organization (O=)
   - Country (C=)

   This creates an unencrypted private key and a matching certificate, both in PEM (Privacy Enhanced Mail) format.

   For simplicity, just enter values for common name, organization, and country. These attributes and values are important when creating a policy.
2. IBM MQ AMS requires that both the certificate and private key are held in the same file. Issue the following command to achieve this:

   ```
   cat $HOME/mycert.pem >> $HOME/private.pem
   ```

   The `private.pem` file in `$HOME` now contains a matching private key and certificate, while the `mycert.pem` file contains all of the public certificates for which you can encrypt messages and validate signatures.

   The two files need to be associated with your environment by creating a keystore configuration file (`keystore.conf`) in your default location.

   By default, IBM MQ AMS looks for the keystore configuration in a `.mqs` subdirectory of your home directory.
3. In QShell create the `keystore.conf` file;

   ```
   mkdir -p $HOME/.mqs
   echo "pem.private = $HOME/private.pem" > $HOME/.mqs/keystore.conf
   echo "pem.public = $HOME/mycert.pem" >> $HOME/.mqs/keystore.conf
   echo "pem.password = unused" >> $HOME/.mqs/keystore.conf
   ```

Creating a policy

Before creating a policy, you need to create a queue to hold protected messages.

Procedure

1. At a command line prompt enter;

   ```
   CRTMQMQ QNAME(PROTECTED) QTYPE(*LCL) MQMNAME (<mqmname>)
   ```

   where `mqmname` is the name of your queue manager.

   Use the DSPMQM command to check that the queue manager is capable of using security policies. Ensure that **`Security Policy Capability`** shows **YES*.

   The simplest policy you can define is an integrity policy, which is achieved by creating a policy with a digital signature algorithm but no encryption algorithm.

   Messages are signed but not encrypted. If messages are to be encrypted, you must specify an encryption algorithm, and one or more intended message recipients.

   A certificate in the public keystore for an intended message recipient is identified through a distinguished name.

2. Display the distinguished names of the certificates in the public keystore, `mycert.pem` in `$HOME`, by using the following command in QShell:

   ```
   /QOpenSys/usr/bin/openssl x509 -in $HOME/mycert.pem -noout -subject -nameopt RFC2253
   ```

   You need to enter the distinguished name as an intended recipient, and the policy name must match the queue name to be protected.
3. At a CL command prompt enter, for example:

   ```
   SETMQMSPL POLICY(PROTECTED) MQMNAME (<mqmname>)SIGNALG(*SHA256) ENCALG(*AES256) RECIP('CN=.., O=.., C=..')
   ```

   where `mqmname` is the name of your queue manager.

   Once the policy is created, any messages that are put, browsed, or destructively removed through that queue name are subject to the IBM MQ AMS policy.

Related information:

Display Message Queue Manager (DSPMQM)

Set MQM Security Policy (SETMQMSPL)

Testing the policy

Use the sample applications provided with the product to test your security policies.

About this task

You can use the sample applications provided with IBM MQ , such as AMQSPUT4, AMQSGET4, AMQSGBR4, and tools such as WRKMQMMSG to put, browse, and get messages using the PROTECTED queue name.

Provided everything has been configured correctly, there should be no difference in application behavior to that of an unprotected queue for this user.

A user not setup for IBM MQ Advanced Message Security , or a user that does not have the required private key to decrypt the message will, however, not be able to view the message. The user receives a completion code of RCFAIL, equivalent to MQCC_FAILED (2) and reason code of RC2063 (MQRC_SECURITY_ERROR).

To see that AMS protection is in effect, put some test messages to the PROTECTED queue, for example using AMQSPUT0. You can then create an alias queue to browse the raw protected data while at rest.

Procedure

To grant necessary permissions to a user, run:

```
CRTMQMQ QNAME(ALIAS) QTYPE(*ALS) TGTQNAME(PROTECTED) MQMNAME(<yourqm>)
```

Browsing using the ALIAS queue name, for example using AMQSBCG4 or WRKMQMMSG, should reveal larger `scrambled` messages where a browse of the PROTECTED queue shows cleartext messages. The scrambled messages are visible, but the original cleartext is not decipherable using the ALIAS queue, as there is no policy for AMS to enforce matching this name. Hence, the raw protected data is returned.

Related information:
Set MQM Security Policy (SETMQMSPL)
Work with MQ Messages (WRKMQMMSG)

Command and configuration events

With IBM MQ Advanced Message Security, you can generate command and configuration event messages, which can be logged and serve as a record of policy changes for auditing.

Command and configuration events generated by IBM MQ are messages of the PCF format sent to dedicated queues on the queue manager where the event occurs.

Configuration events messages are sent to the SYSTEM.ADMIN.CONFIG.EVENT queue.

Command events messages are sent to the SYSTEM.ADMIN.COMMAND.EVENT queue.

Events are generated regardless of tools you are using to manage IBM MQ Advanced Message Security security policies.

In IBM MQ Advanced Message Security, there are four types of events generated by different actions on security policies:

- "Creating security policies" on page 902, which generate two IBM MQ event messages:
  - A configuration event
  - A command event
- "Changing security policies" on page 903, which generates three IBM MQ event messages:
  - A configuration event that contains old security policy values
  - A configuration event that contains new security policy values
  - A command event
- "Displaying and dumping security policies" on page 903, which generates one IBM MQ event message:
  - A command event
- "Removing security policies" on page 905, which generates two IBM MQ event messages:
  - A configuration event
  - A command event

Enabling and disabling event logging

You control command and configuration events by using the queue manager attributes CONFIGEV and CMDEV. To enable these events, set the appropriate queue manager attribute to `ENABLED`. To disable these events, set the appropriate queue manager attribute to `DISABLED`.

Procedure

Configuration events

To enable configuration events, set CONFIGEV to `ENABLED`. To disable configuration events, set CONFIGEV to `DISABLED`. For example, you can enable configuration events by using the following MQSC command:

```
ALTER QMGR CONFIGEV (ENABLED)
```

Command events

To enable command events, set CMDEV to `ENABLED`. To enable command events for commands except DISPLAY MQSC commands and Inquire PCF commands, set the CMDEV to NODISPLAY. To disable command events, set CMDEV to `DISABLED`. For example, you can enable command events by using the following MQSC command:

```
ALTER QMGR CMDEV (ENABLED)
```

Related information:

Controlling configuration, command, and logger events in Websphere MQ

Command event message format

Command event message consists of MQCFH structure and PCF parameters following it.

Here are selected MQCFH values:

```
Type = MQCFT_EVENT;
Command = MQCMD_COMMAND_EVENT;
MsgSeqNumber = 1;
Control = MQCFC_LAST;
ParameterCount = 2;
CompCode = MQCC_WARNING;
Reason = MQRC_COMMAND_PCF;
```

Note: ParameterCount value is two because there are always two paramteters of MQCFGR type (group). Each group consists of appropriate parameters. The event data consists of two groups, CommandContext and CommandData.

CommandContext contains:

EventUserID

Description:	The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier:	MQCACF_EVENT_USER_ID.
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

EventOrigin

Description:	The origin of the action causing the event.
Identifier:	MQIACF_EVENT_ORIGIN.
Data type:	MQCFIN.
Values:	**MQEVO_CONSOLE** Console command - command line. **MQEVO_MSG** Command message from IBM MQ Explorer plugin.
Returned:	Always.

EventQMgr

Description:	The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier:	MQCACF_EVENT_Q_MGR.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

EventAccountingToken

Description:	For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MD of the command message.
Identifier:	MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type:	MQCFBS.
Maximum length:	MQ_ACCOUNTING_TOKEN_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventIdentityData

Description:	For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MD of the command message.
Identifier:	MQCACF_EVENT_APPL_IDENTITY.
Data type:	MQCFST.
Maximum length:	MQ_APPL_IDENTITY_DATA_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventApplType

Description:	For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MD of the command message.
Identifier:	MQIACF_EVENT_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventApplName

Description:	For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MD of the command message.
Identifier:	MQCACF_EVENT_APPL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description:	For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MD of the command message.
Identifier:	MQCACF_EVENT_APPL_ORIGIN.
Data type:	MQCFST.
Maximum length:	MQ_APPL_ORIGIN_DATA_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

Command

Description:	The command code.
Identifier:	MQIACF_COMMAND.
Data type:	MQCFIN.
Values:	**MQCMD_INQUIRE_PROT_POLICY numeric value 205** **MQCMD_CREATE_PROT_POLICY numeric value 206** **MQCMD_DELETE_PROT_POLICY numeric value 207** **MQCMD_CHANGE_PROT_POLICY numeric value 208** These are defined in IBM MQ 8.0 `cmqcfc.h`
Returned:	Always.

CommandData contains PCF elements that comprised the PCF command.

Configuration event message format

Configuration events are PCF messages of standard IBM MQ Advanced Message Security format.

Possible values for the MQMD message descriptor can be found in Event message MQMD (message descriptor).

Here are selected MQMD values:

```
Format = MQFMT_EVENT
Peristence = MQPER_PERSISTENCE_AS_Q_DEF
PutApplType = MQAT_QMGR           //for both CLI and command server
```

Message buffer consist of MQCFH structure and the parameter structure that follows it. Possible MQCFH values can be found in Event message MQCFH (PCF header).

Here are selected MQCFH values:

```
Type = MQCFT_EVENT
Command = MQCMD_CONFIG_EVENT
MsgSeqNumber = 1 or 2          // 2 will be in case of Change Object event
Control = MQCFC_LAST or MQCFC_NOT_LAST     //MQCFC_NOT_LAST will be in case of 1 Change Object event
ParameterCount = reflects number of PCF parameters following MQCFH
CompCode = MQCC_WARNING
Reason = one of {MQRC_CONFIG_CREATE_OBJECT, MQRC_CONFIG_CHANGE_OBJECT, MQRC_CONFIG_DELETE_OBJECT}
```

The parameters following MQCFH are:

EventUserID

Description:	The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier:	**MQCACF_EVENT_USER_ID**
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

SecurityId

Description:	Value of MQMD.AccountingToken in case of command server message or Windows SID for local command.
Identifier:	**MQBACF_EVENT_SECURITY_ID**
Data type:	MQCBS.
Maximum length:	MQ_SECURITY_ID_LENGTH.
Returned:	Always.

EventOrigin

Description:	The origin of the action causing the event.
Identifier:	**MQIACF_EVENT_ORIGIN**
Data type:	MQCFIN.
Values:	**MQEVO_CONSOLE** Console command - command line. **MQEVO_MSG** Command message from the IBM MQ Explorer plugin.
Returned:	Always.

EventQMgr

Description:	The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier:	**MQCACF_EVENT_Q_MGR**
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

ObjectType

Description:	Object type.
Identifier:	**MQIACF_OBJECT_TYPE**
Data type:	MQCFIN
Value:	**MQOT_PROT_POLICY** IBM MQ Advanced Message Security protection policy. **1019** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Always.

PolicyName

Description:	The IBM MQ Advanced Message Security policy name.
Identifier:	**MQCA_POLICY_NAME**.
Data type:	MQCFST.
Value:	**2112** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Maximum length:	MQ_OBJECT_NAME_LENGTH.
Returned:	Always.

PolicyVersion

Description:	The IBM MQ Advanced Message Security policy version.
Identifier:	**MQIA_POLICY_VERSION**
Data type:	MQCFIN
Value	**238** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Always

TolerateFlag

Description:	The IBM MQ Advanced Message Security policy toleration flag.
Identifier:	**MQIA_TOLERATE_UNPROTECTED**
Data type:	MQCFIN
Value	**235** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Always.

SignatureAlgorithm

Description:	The IBM MQ Advanced Message Security policy signature algorithm.
Identifier:	**MQIA_SIGNATURE_ALGORITHM**
Data type:	MQCFIN
Value:	**236** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Whenever there is a signature algorithm defined in the IBM MQ Advanced Message Security policy

EncryptionAlgorithm

Description:	The IBM MQ Advanced Message Security policy encryption algorithm.
Identifier:	**MQIA_ENCRYPTION_ALGORITHM**
Data type:	MQCFIN
Value:	**237** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Whenever there is an encryption algorithm defined in the IBM MQ policy

SignerDNs

Description:	Subject DistinguishedName of allowed signers.
Identifier:	**MQCA_SIGNER_DN**
Data type:	MQCFSL
Value:	**2113** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Maximum length:	Longest signer DN in the policy, but no longer then MQ_DISTINGUISHED_NAME_LENGTH
Returned:	Whenever defined in IBM MQ policy.

RecipientDNs

Description:	Subject DistinguishedName of allowed signers.
Identifier:	**MQCA_RECIPIENT_DN**
Data type:	MQCFSL
Value:	**2114** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Maximum length:	Longest recipient DN in the policy, but no longer then MQ_DISTINGUISHED_NAME_LENGTH.
Returned:	Whenever defined in IBM MQ policy.

Problems and solutions

Information is provided to help you identify and resolve problems relating to IBM MQ Advanced Message Security.

For problems relating to IBM MQ Advanced Message Security check the queue manager error log first.

com.ibm.security.pkcsutil.PKCSException: Error encrypting contents

Error `com.ibm.security.pkcsutil.PKCSException: Error encrypting contents` suggests that IBM MQ Advanced Message Security has problems with accessing cryptographic algorithms.

If the following error is returned by IBM MQ Advanced Message Security:

```
DRQJP0103E  The IBM MQ Advanced Message Security Java interceptor failed to protect message.
com.ibm.security.pkcsutil.PKCSException: Error encrypting contents (java.security.InvalidKeyException: Illegal key size
```

verify if the JCE security policy in `JAVA_HOME/lib/security/local_policy.jar/*.policy` grants access to the signature algorithms used in MQ AMS policy.

If the signature algorithm you want to use is not specified in your current security policy, download the correct Java policy file, for your version of the product, from the following location :Fixes in the IBM Developer Kits.

OSGi support

To use OSGi bundle with IBM MQ Advanced Message Security additional parameters are required.

Run the following parameter during the OSGi bundle startup:

```
-Dorg.osgi.framework.system.packages.extra=com.ibm.security.pkcs7
```

When using encrypted password in your keystore.conf, the following statement must be added when OSGi bundle is running:

```
-Dorg.osgi.framework.system.packages.extra=com.ibm.security.pkcs7,com.ibm.misc
```

Restriction: IBM MQ AMS supports communication using only MQ Base Java Classes for queues protected from within the OSGi bundle.

Problems opening protected queues when using JMS

Various problems can arise when you open protected queues when using IBM MQ Advanced Message Security.

You are running JMS and you receive error 2085 (MQRC_UNKNOWN_OBJECT_NAME) together with error JMSMQ2008.

You have verified that you have set up your IBM MQ AMS as described in "Quick Start Guide for IBM MQ AMS with Java clients" on page 863.

A possible cause is that you are using a non-IBM Java Runtime Environment. This is a known limitation described in "Known limitations" on page 852.

You have not set the AMQ_DISABLE_CLIENT_AMS environment variable.

Resolving the problem

There are four options for working around this problem:

1. Start your JMS application under a supported IBM Java Runtime Environment (JRE).

2. Move your application to the same machine where your queue manager is running and have it connect using a bindings mode connection.

   A bindings mode connection uses platform native libraries to perform the IBM MQ API calls. Accordingly, the native AMS interceptor is used to perform the AMS operations and there is no reliance on the capabilities of the JRE.
3. Use an MCA interceptor, because this allows signing and encryption of messages as soon as they arrive at the queue manager, without the need for the client to perform any AMS processing.

   Given that the protection is applied at the queue manager, an alternate mechanism must be used to protect the messages in transit from the client to the queue manager. Most commonly this is achieved by configuring SSL/TLS encryption on the server connection channel used by the application.
4. Set the AMQ_DISABLE_CLIENT_AMS environment variable if you do not want to use IBM MQ AMS.

See "Message Channel Agent (MCA) interception" on page 878 for further information.

Note: A security policy must be in place for each queue that the MCA Interceptor will deliver messages onto. In other words, the target queue needs to have an AMS security policy in place with the distinguished name (DN) of the signer and recipient matching that of the certificate assigned to the MCA Interceptor. That is, the DN of the certificate designated by `cms.certificate.channel.SYSTEM.DEF.SVRCONN` property in the `keystore.conf` used by the queue manager.

Example configurations on z/OS

This section provides example configurations of policies and certificates for IBM MQ Advanced Message Security queuing scenarios on z/OS.

The examples cover the IBM MQ Advanced Message Security policies required, and the digital certificates that must exist relative to users and key rings. The examples assume that the users involved in the scenarios have been set up by following the instructions provided in Task 27: Grant users resource permissions for IBM MQ Advanced Message Security.

Local queuing of integrity-protected messages on z/OS

This example details the IBM MQ Advanced Message Security policies and certificates needed to send and retrieve integrity-protected messages to and from a queue, local to the putting and getting applications.

The example queue manager and queue are:

```
BNK6        - Queue manager
FIN.XFER.Q7 - Local queue
```

These users are used:

```
WMQBNK6  - IBM MQ AMS task user
TELLER5  - Sending user
FINADM2  - Recipient user
```

Create the user certificates

In this example, only one user certificate is needed. This is the sending user's certificate which is needed to sign integrity-protected messages. The sending user is 'TELLER5'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the IBM MQ Advanced Message Security task user, in this case user WMQBNK6.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue a user certificate for user 'TELLER5'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, IBM MQ Advanced Message Security requires:
- The CA certificate (chain).
- The user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set. For more information about these and other RACDCERT commands, refer to *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case, are required on the z/OS system running queue manager BNK6.

When the certificates have been imported on the z/OS system running BNK6, the user certificate requires the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
```

In this example, no certificate is required for the recipient user.

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS system running BNK6. To create the key rings use the RACDCERT ADDRING commands:

```
RACDCERT ID(WMQBNK6) ADDRING(drq.ams.keyring)
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
```

This creates a key ring for the IBM MQ Advanced Message Security task user, WMQBNK6, and a key ring for the sending user, 'TELLER5'. Note that the key ring name drq.ams.keyring is mandatory, and the name is case-sensitive.

When the key rings have been created, the relevant certificates can be connected:

```
RACDCERT ID(WMQBNK6) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))

RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

The sending user certificate must be connected as DEFAULT. If the sending user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing purposes.

The creation and modification of certificates is not recognized by IBM MQ Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **MODIFY** command is used to refresh the IBM MQ Advanced Message Security certificate configuration. For example:

```
F BNK6AMSM,REFRESH KEYRING
```

Create the IBM MQ Advanced Message Security policy

In this example, integrity-protected messages are put to queue FIN.XFER.Q7 by an application running as user 'TELLER5', and retrieved from the same queue by an application running as user 'FINADM2', so only one IBM MQ Advanced Message Security policy is required.

IBM MQ Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command:

```
setmqspl -m BNK6 -p FIN.XFER.Q7 -s MD5 -a CN=Teller5,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q7. The algorithm that is used to generate the sender's signature is MD5, and the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US'.

After defining the policy, either restart the BNK6 queue manager, or use the z/OS **MODIFY** command to refresh the IBM MQ Advanced Message Security policy configuration. For example:

```
F BNK6AMSM,REFRESH POLICY
```

Local queuing of privacy-protected messages on z/OS

This example details the IBM MQ Advanced Message Security policies and certificates needed to send and retrieve privacy-protected messages to and from a queue, local to the putting and getting applications. Privacy-protected messages are both signed and encrypted.

The example queue manager and local queue are as follows:

```
BNK6        - Queue manager
FIN.XFER.Q8 - Local queue
```

These users are used:

```
WMQBNK6  - IBM MQ AMS task user
TELLER5  - Sending user
FINADM2  - Recipient user
```

The steps to configure this scenario are:

Create the user certificates

In this example, two user certificates are required. These are the sending user's certificate which is needed to sign messages, and the recipient user's certificate which is needed to encrypt and decrypt the message data. The sending user is 'TELLER5' and the recipient user is 'FINADM2'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the IBM MQ Advanced Message Security task user, in this case user WMQBNK6.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue user certificates for users 'TELLER5' and 'FINADM2'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)

RACDCERT ID(FINADM2) GENCERT SUBJECTSDN(CN('FinAdm2') O('BCO') C('US'))
WITHLABEL('FinAdm2') SIGNWITH(CERTAUTH LABEL('BCOCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, IBM MQ Advanced Message Security requires:
- The CA certificate (chain).
- The sending user certificate and its private key.
- The recipient user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set. For more information about these and other RACDCERT commands, refer to *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case are required on the z/OS system running queue manager BNK6.

When the certificates have been imported on the z/OS system running BNK6, the user certificates require the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
RACDCERT ID(FINADM2) ALTER (LABEL('FinAdm2')) TRUST
```

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS system running BNK6. To create the key rings use the RACDCERT ADDRING command:

```
RACDCERT ID(WMQBNK6) ADDRING(drq.ams.keyring)
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
RACDCERT ID(FINADM2) ADDRING(drq.ams.keyring)
```

This creates a key ring for the IBM MQ Advanced Message Security task user and key rings for the sending and recipient users. Note that the key ring name drq.ams.keyring is mandatory, and the name is case-sensitive.

When the key rings have been created, the relevant certificates can be connected.

```
RACDCERT ID(WMQBNK6) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))

RACDCERT ID(WMQBNK6) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) USAGE(SITE))

RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))

RACDCERT ID(FINADM2) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

The sending and recipient user certificates must be connected as DEFAULT. If either user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing and decryption purposes.

The recipient user's certificate must also be connected to the IBM MQ Advanced Message Security task user's key ring with USAGE(SITE). This is because the Advanced Message Security task needs the recipient's public key when encrypting the message data. The USAGE(SITE) prevents the private key from being accessible in the key ring.

The creation and modification of certificates is not recognized by IBM MQ Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **MODIFY** command is used to refresh the IBM MQ Advanced Message Security certificate configuration. For example:

```
F BNK6AMSM,REFRESH KEYRING
```

Create the IBM MQ Advanced Message Security policy

In this example, privacy-protected messages are put to queue FIN.XFER.Q8 by an application running as user 'TELLER5', and retrieved from the same queue by an application running as user 'FINADM2', so only one IBM MQ Advanced Message Security policy is required.

IBM MQ Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command:

```
setmqspl -m BNK6 -p FIN.XFER.Q8 -s SHA1 -e 3DES -a CN=Teller5,O=BCO,C=US -r CN=FinAdm2,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q8. The algorithm that is used to generate the sender's signature is SHA1, and the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US', and the recipient user is 'CN=FinAdm2,O=BCO,C=US'. The algorithm that is used to encrypt the message data is 3DES.

After defining the policy, either restart the BNK6 queue manager, or use the z/OS **MODIFY** command to refresh the IBM MQ Advanced Message Security policy configuration. For example:

```
F BNK6AMSM,REFRESH POLICY
```

Remote queuing of integrity-protected messages on z/OS

This example details the IBM MQ Advanced Message Security policies and certificates needed to send and retrieve integrity-protected messages to and from queues managed by two different queue managers. The two queue managers can be running on the same z/OS system, or on different z/OS systems, or one queue manager can be on a distributed system running IBM MQ Advanced Message Security.

The example queue managers and queues are:

```
BNK6        - Sending queue manager
BNK7        - Recipient queue manager
FIN.XFER.Q7 - Remote queue on BNK6
FIN.RCPT.Q7 - Local queue on BNK7
```

Note: For this example, BNK6 and BNK7 are queue managers running on different z/OS systems.

These users are used:

```
WMQBNK6  - IBM MQ AMS task user on BNK6
WMQBNK7  - IBM MQ AMS task user on BNK7
TELLER5  - Sending user on BNK6
FINADM2  - Recipient user on BNK7
```

The steps to configure this scenario are as follows:

Create the user certificates

In this example, only one user certificate is needed. This is the sending user's certificate which is needed to sign integrity-protected message. The sending user is 'TELLER5'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the IBM MQ Advanced Message Security task user, in this case user WMQBNK7.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue user certificate for user 'TELLER5'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, IBM MQ Advanced Message Security require:
- The CA certificate (chain).
- The sending user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set. For more information about these and other RACDCERT commands, refer to *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case, are required on the z/OS system running queue manager BNK6 and BNK7.

In this example, the sending certificate must be imported on the z/OS system running BNK6, and the CA certificate must be imported on the z/OS system running BNK7. When the certificates have been imported, the user certificate requires the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example, on BNK6:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
```

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS system running BNK6 and BNK7.

To create the key rings use the RACDCERT ADDRING command, on BNK6:

```
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
```

This creates a key ring for the sending user on BNK6. Note that the key ring name drq.ams.keyring is mandatory, and the name is case-sensitive.

On BNK7:

```
RACDCERT ID(WMQBNK7) ADDRING(drq.ams.keyring)
```

This creates a key ring for the IBM MQ Advanced Message Security task user on BNK7. No user key ring is required for 'TELLER5' on BNK7.

When the key rings have been created, the relevant certificates can be connected.

On BNK6:

```
RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

On BNK7:

```
RACDCERT ID(WMQBNK7) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))
```

The sending user certificate must be connected as DEFAULT. If the sending user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing purposes.

The creation and modification of certificates is not recognized by IBM MQ Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **MODIFY** command is used to refresh the IBM MQ Advanced Message Security certificate configuration. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,KEYRING
```

On BNK7:

```
F BNK7AMSM,REFRESH,KEYRING
```

Create the IBM MQ Advanced Message Security policies

In this example, integrity-protected messages are put to remote queue FIN.XFER.Q7 on BNK6 by an application running as user 'TELLER5', and retrieved from local queue FIN.RCPT.Q7 on BNK7 by an application running as user 'FINADM2', so two IBM MQ Advanced Message Security policies are required.

IBM MQ Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command to define an integrity policy for the remote queue on BNK6:

```
setmqspl -m BNK6 -p FIN.XFER.Q7 -s MD5 -a CN=Teller5,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q7. The algorithm that is used to generate the sender's signature is MD5, and the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US'.

Also, use the CSQ0UTIL utility to run the following command to define an integrity policy for the local queue on BNK7:

```
setmqspl -m BNK7 -p FIN.RCPT.Q7 -s MD5 -a CN=Teller5,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK7. The policy name and associated queue is FIN.RCPT.Q7. The algorithm expected for the sender's signature is MD5, and the distinguished name (DN) of the sending user is expected to be 'CN=Teller5,O=BCO,C=US'.

After defining the two policies, either restart the BNK6 and BNK7 queue managers, or use the z/OS **MODIFY** command to refresh the IBM MQ Advanced Message Security policy configurations. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,POLICY
```

On BNK7:

```
F BNK7AMSM,REFRESH,POLICY
```

Remote queuing of privacy-protected messages on z/OS

This example details the IBM MQ Advanced Message Security policies and certificates needed to send and retrieve privacy-protected messages to and from queues managed by two different queue managers. The two queue managers can be running on the same z/OS system, or on different z/OS systems, or one queue manager can be on a distributed system running IBM MQ Advanced Message Security.

The example queue managers and queues are:

```
BNK6        - Sending queue manager
BNK7        - Recipient queue manager
FIN.XFER.Q7 - Remote queue on BNK6
FIN.RCPT.Q7 - Local queue on BNK7
```

Note: For this example, BNK6 and BNK7 are queue managers running on different z/OS systems of the same name.

These users are used:

```
WMQBNK6  - IBM MQ AMS task user on BNK6
WMQBNK7  - IBM MQ AMS task user on BNK7
TELLER5  - Sending user on BNK6
FINADM2  - Recipient user on BNK7
```

The steps to configure this scenario are as follows:

Create the user certificates

In this example, two user certificates are required. These are the sending user's certificate which is needed to sign messages, and the recipient user's certificate which is needed to encrypt and decrypt the message data. The sending user is 'TELLER5' and the recipient user is 'FINADM2'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the IBM MQ Advanced Message Security task user, in this case user WMQBNK7.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue user certificates for users 'TELLER5' and 'FINADM2'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)

RACDCERT ID(FINADM2) GENCERT SUBJECTSDN(CN('FinAdm2') O('BCO') C('US'))
WITHLABEL('FinAdm2') SIGNWITH(CERTAUTH LABEL('BCOCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, IBM MQ Advanced Message Security requires:
- The CA certificate (chain).
- The sending user certificate and its private key.
- The recipient user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set. For more information about these and other RACDCERT commands, refer to *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case, are required on the z/OS system running queue manager BNK6 and BNK7.

In this example, the sending and recipient certificates must be imported on the z/OS system running BNK6, and the CA and recipient certificates must be imported on the z/OS system running BNK7. When the certificates have been imported, the user certificates require the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example:

On BNK6:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
RACDCERT ID(FINADM2) ALTER (LABEL('FinAdm2')) TRUST
```

On BNK7:

```
RACDCERT ID(FINADM2) ALTER (LABEL('FinAdm2')) TRUST
```

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS systems running BNK6 and BNK7.

To create the key rings use the RACDCERT ADDRING command:

On BNK6:

```
RACDCERT ID(WMQBNK6) ADDRING(drq.ams.keyring)
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
```

This creates a key ring for the IBM MQ Advanced Message Security task user and a key ring for the sending user on BNK6. Note that the key ring name drq.ams.keyring is mandatory, and the name is case-sensitive.

On BNK7:

```
RACDCERT ID(WMQBNK7) ADDRING(drq.ams.keyring)
RACDCERT ID(FINADM2) ADDRING(drq.ams.keyring)
```

This creates a key ring for the IBM MQ Advanced Message Security task user and a key ring for the recipient user on BNK7.

When the key rings have been created, the relevant certificates can be connected.

On BNK6:

```
RACDCERT ID(WMQBNK6) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) USAGE(SITE))

RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

On BNK7:

```
RACDCERT ID(WMQBNK7) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))

RACDCERT ID(FINADM2) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

The sending and recipient user certificates must be connected as DEFAULT. If either user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing and encryption/decryption purposes.

On BNK6, the recipient user's certificate must also be connected to the IBM MQ Advanced Message Security task user's key ring with USAGE(SITE). This is because the Advanced Message Security task needs the recipient's public key when encrypting the message data. The USAGE(SITE) prevents the private key from being accessible in the key ring.

The creation and modification of certificates is not recognized by IBM MQ Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **MODIFY** command is used to refresh the IBM MQ Advanced Message Security certificate configuration. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,KEYRING
```

On BNK7:

```
F BNK7AMSM,REFRESH,KEYRING
```

Create the IBM MQ Advanced Message Security policies

In this example, privacy-protected messages are put to remote queue FIN.XFER.Q7 on BNK6 by an application running as user 'TELLER5', and retrieved from local queue FIN.RCPT.Q7 on BNK7 by an application running as user 'FINADM2', so two IBM MQ Advanced Message Security policies are required.

IBM MQ Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command to define a privacy policy for the remote queue on BNK6:

```
setmqspl -m BNK6 -p FIN.XFER.Q7 -s SHA1 -e 3DES -a CN=Teller5,O=BCO,C=US -r CN=FinAdm2,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q7. The algorithm that is used to generate the sender's signature is SHA1, the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US', and the recipient user is 'CN=FinAdm2,O=BCO,C=US'. The algorithm that is used to encrypt the message data is 3DES.

Also, use the CSQ0UTIL utility to run the following command to define a privacy policy for the local queue on BNK7:

```
setmqspl -m BNK7 -p FIN.RCPT.Q7 -s SHA1 -e 3DES -a CN=Teller5,O=BCO,C=US -r CN=FinAdm2,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK7. The policy name and associated queue is FIN.RCPT.Q7. The algorithm expected for the sender's signature is SHA1, the distinguished name (DN) of the sending user is expected to be 'CN=Teller5,O=BCO,C=US', and the recipient user is 'CN=FinAdm2,O=BCO,C=US'. The algorithm that is used to decrypt the message data is 3DES.

After defining the two policies, either restart the BNK6 and BNK7 queue managers, or use the z/OS **MODIFY** command to refresh the IBM MQ Advanced Message Security policy configuration. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,POLICY
```

On BNK7:

```
F BNK7AMSM,REFRESH,POLICY
```

Monitoring and performance

Use the monitoring information and guidance in this section, and the specific tuning tips, to help improve the performance of your queue manager network.

Depending on the size and complexity of your queue manager network, you can obtain a range of information from monitoring the network. You can use that information, along with the information provided in specific tuning tips, to help you tune your network performance.

Monitoring your IBM MQ network

A number of monitoring techniques are available in IBM MQ to obtain statistics and other specific information about how your queue manager network is running. Use the monitoring information and guidance in this section to help improve the performance of your queue manager network.

The following list provides examples of reasons for monitoring your queue manager network:

- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm that your queue manager network is running correctly.
- Generate messages when certain events occur.
- Record message activity.
- Determine the last known location of a message.
- Check various statistics of a queue manager network in real time.
- Generate an audit trail.
- Account for application resource usage.
- Capacity planning.

Event monitoring

Event monitoring is the process of detecting occurrences of *instrumentation events* in a queue manager network. An instrumentation event is a logical combination of events that is detected by a queue manager or channel instance. Such an event causes the queue manager or channel instance to put a special message, called an *event message*, on an event queue.

IBM MQ instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. Use these events to monitor the operation of the queue managers in your queue manager network to achieve the following goals:

- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Generate an audit trail.
- React to queue manager state changes

Related reference:

"Event types" on page 932
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information:

Event message reference

Event message format

Instrumentation events

An instrumentation event is a logical combination of conditions that a queue manager or channel instance detects and puts a special message, called an *event message*, on an event queue.

IBM MQ instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. You can use these events to monitor the operation of queue managers (with other methods such as Tivoli NetView for z/OS).

Figure 86 on page 931 illustrates the concept of instrumentation events.

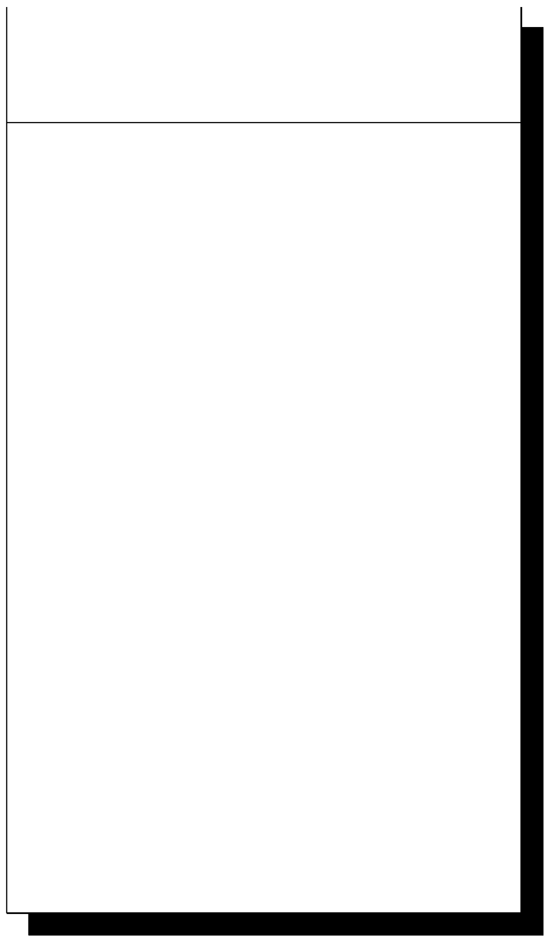


Figure 86. Understanding instrumentation events

Event monitoring applications

Applications that use events to monitor queue managers must include the following provisions:

1. Set up channels between the queue managers in your network.
2. Implement the required data conversions. The normal rules of data conversion apply. For example, if you are monitoring events on a UNIX system queue manager from a z/OS queue manager, ensure that you convert EBCDIC to ASCII.

Event notification through event queues

When an event occurs, the queue manager puts an event message on the appropriate event queue, if defined. The event message contains information about the event that you can retrieve by writing a suitable MQI application program that performs the following steps:

- Get the message from the queue.
- Process the message to extract the event data.

The related information describes the format of event messages.

Conditions that cause events

The following list gives examples of conditions that can cause instrumentation events:

- A threshold limit for the number of messages on a queue is reached.
- A channel instance is started or stopped.
- A queue manager becomes active, or is requested to stop.
- An application tries to open a queue specifying a user ID that is not authorized on IBM MQ for IBM i, Windows, UNIX and Linux systems.
- Objects are created, deleted, changed, or refreshed.
- An MQSC or PCF command runs successfully.
- A queue manager starts writing to a new log extent.
- Putting a message on the dead-letter queue, if the event conditions are met.

Related concepts:

"Performance events" on page 944
Performance events relate to conditions that can affect the performance of applications that use a specified queue. The scope of performance events is the queue. **MQPUT** calls and **MQGET** calls on one queue do not affect the generation of performance events on another queue.

"Sample program to monitor instrumentation events" on page 977
V 8.0.0.4 **amqsevt** formats the instrumentation events that a queue manager can create, and is supplied with IBM MQ. The program reads messages from event queues, and formats them into readable strings.

Event types:

Use this page to view the types of instrumentation event that a queue manager or channel instance can report

IBM MQ instrumentation events have the following types:

- Queue manager events
- Channel and bridge events
- Performance events
- Configuration events
- Command events
- Logger events
- Local events

For each queue manager, each category of event has its own event queue. All events in that category result in an event message being put onto the same queue.

This event queue:	Contains messages from:
SYSTEM.ADMIN.QMGR.EVENT	Queue manager events
SYSTEM.ADMIN.CHANNEL.EVENT	Channel events
SYSTEM.ADMIN.PERFM.EVENT	Performance events
SYSTEM.ADMIN.CONFIG.EVENT	Configuration events
SYSTEM.ADMIN.COMMAND.EVENT	Command events
SYSTEM.ADMIN.LOGGER.EVENT	Logger events
SYSTEM.ADMIN.PUBSUB.EVENT	Gets events related to Publish/Subscribe. Only used with Multicast. For more information see, Multicast application monitoring.

By incorporating instrumentation events into your own system management application, you can monitor the activities across many queue managers, across many different nodes, and for multiple IBM MQ applications. In particular, you can monitor all the nodes in your system from a single node (for those nodes that support IBM MQ events) as shown inFigure 87.

Instrumentation events can be reported through a user-written reporting mechanism to an administration application that can present the events to an operator.

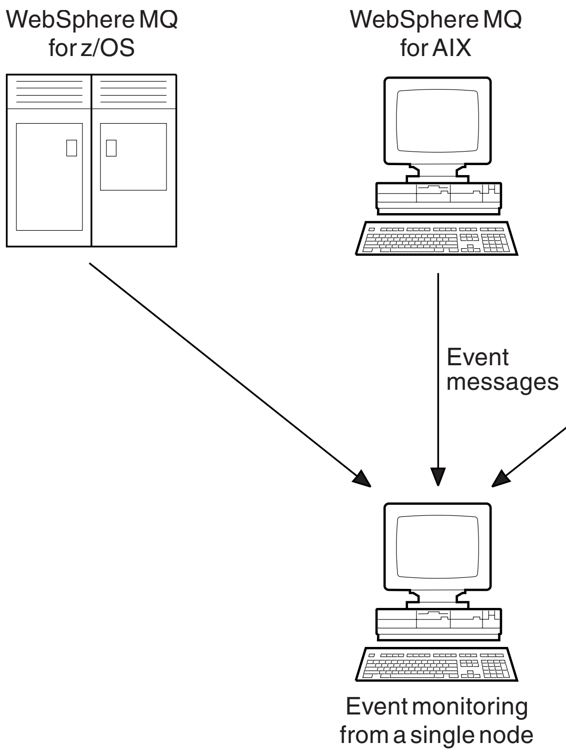


Figure 87. Monitoring queue managers across different platforms, on a single node

Instrumentation events also enable applications acting as agents for other administration networks, for example Tivoli NetView for z/OS, to monitor reports and create the appropriate alerts.

Queue manager events:

Queue manager events are related to the use of resources within queue managers. For example, a queue manager event is generated if an application tries to put a message on a queue that does not exist.

The following examples are conditions that can cause a queue manager event:

- An application issues an MQI call that fails. The reason code from the call is the same as the reason code in the event message.

  A similar condition can occur during the internal operation of a queue manager; for example, when generating a report message. The reason code in an event message might match an MQI reason code, even though it is not associated with any application. Do not assume that, because an event message reason code looks like an MQI reason code, the event was necessarily caused by an unsuccessful MQI call from an application.
- A command is issued to a queue manager and processing this command causes an event. For example:
  - A queue manager is stopped or started.
  - A command is issued where the associated user ID is not authorized for that command.

IBM MQ puts messages for queue manager events on the SYSTEM.ADMIN.QMGR.EVENT queue, and supports the following queue manager event types:

Authority (on Windows, and UNIX systems only)

Authority events report an authorization, such as an application trying to open a queue for which it does not have the required authority, or a command being issued from a user ID that does not have the required authority. The authority event message can contain the following event data:

- Not Authorized (type 1)
- Not Authorized (type 2)
- Not Authorized (type 3)
- Not Authorized (type 4)
- Not Authorized (type 5)
- Not Authorized (type 6)

All authority events are valid on Windows, and UNIX systems only.

Inhibit

Inhibit events indicate that an MQPUT or MQGET operation has been attempted against a queue where the queue is inhibited for puts or gets, or against a topic where the topic is inhibited for publishes. The inhibit event message can contain the following event data:

- Get Inhibited
- Put Inhibited

Local

Local events indicate that an application (or the queue manager) has not been able to access a local queue or other local object. For example, an application might try to access an object that has not been defined. The local event message can contain the following event data:

- Alias Base Queue Type Error
- Unknown Alias Base Queue
- Unknown Object Name

Remote

Remote events indicate that an application or the queue manager cannot access a remote queue on another queue manager. For example, the transmission queue to be used might not be correctly defined. The remote event message can contain the following event data:

- Default Transmission Queue Type Error
- Default Transmission Queue Usage Error
- Queue Type Error
- Remote Queue Name Error
- Transmission Queue Type Error
- Transmission Queue Usage Error
- Unknown Default Transmission Queue
- Unknown Remote Queue Manager
- Unknown Transmission Queue

Start and stop

Start and stop events indicate that a queue manager has been started or has been requested to stop or quiesce.

z/OS supports only start events.

Stop events are not recorded unless the default message-persistence of the SYSTEM.ADMIN.QMGR.EVENT queue is defined as persistent. The start and stop event message can contain the following event data:

- Queue Manager Active

- Queue Manager Not Active

For each event type in this list, you can set a queue manager attribute to enable or disable the event type.

Channel and bridge events:

Channels report these events as a result of conditions detected during their operation. For example, when a channel instance is stopped.

Channel events are generated in the following circumstances:
- When a command starts or stops a channel.
- When a channel instance starts or stops.
- When a channel receives a conversion error warning when getting a message.
- When an attempt is made to create a channel automatically; the event is generated whether the attempt succeeds or fails.

Note: Client connections do not cause Channel Started or Channel Stopped events.

When a command is used to start a channel, an event is generated. Another event is generated when the channel instance starts. However, starting a channel by a listener, the **runmqchl** command, or a queue manager trigger message does not generate an event. In these cases, an event is generated only when the channel instance starts.

A successful start or stop channel command generates at least two events. These events are generated for both queue managers connected by the channel (providing they support events).

If a channel event is put on an event queue, an error condition causes the queue manager to create an event.

The event messages for channel and bridge events are put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The channel event messages can contain the following event data:
- Channel Activated
- Channel Auto-definition Error
- Channel Auto-definition OK
- Channel Conversion Error
- Channel Not Activated
- Channel Started
- Channel Stopped
- Channel Stopped By User
- Channel Blocked

z/OS

IMS bridge events (z/OS only)

These events are reported when an IMS bridge starts or stops.

The IMS bridge event messages can contain the following event data:
- Bridge Started
- Bridge Stopped

SSL events

The only Secure Sockets Layer (SSL or TLS) event is the Channel SSL Error event. This event is reported when a channel using SSL or TLS fails to establish an SSL connection.

The SSL event messages can contain the following event data:
- Channel SSL Error
- Channel SSL Warning

Performance events:

Performance events are notifications that a resource has reached a threshold condition. For example, a queue depth limit has been reached.

Performance events relate to conditions that can affect the performance of applications that use a specified queue. They are not generated for the event queues themselves.

The event type is returned in the command identifier field in the message data.

If a queue manager tries to put a queue manager event or performance event message on an event queue and an error that would typically create an event is detected, another event is not created and no action is taken.

MQGET and MQPUT calls within a unit of work can generate performance events regardless of whether the unit of work is committed or backed out.

The event messages for performance events are put on the SYSTEM.ADMIN.PERFM.EVENT queue.

There are two types of performance event:

Queue depth events
Queue depth events relate to the number of messages on a queue; that is, how full or empty the queue is. These events are supported for shared queues. The queue depth event messages can contain the following event data:
- Queue Depth High
- Queue Depth Low
- Queue Full

Queue service interval events
Queue service interval events relate to whether messages are processed within a user-specified time interval. These events are not supported for shared queues.

z/OS IBM MQ for z/OS supports queue depth events for QSGDISP (SHARED) queues, but not service interval events. Queue manager and channel events remain unaffected by shared queues. The queue service event messages can contain the following event data:
- Queue Service Interval High
- Queue Service Interval OK

Configuration events:

Configuration events are generated when a configuration event is requested explicitly, or automatically when an object is created, modified, or deleted.

A configuration event message contains information about the attributes of an object. For example, a configuration event message is generated if a namelist object is created, and contains information about the attributes of the namelist object.

The event messages for configuration events are put on the SYSTEM.ADMIN.CONFIG.EVENT queue.

There are four types of configuration event:

Create object events
Create object events are generated when an object is created. The event message contains the following event data: Create object.

Change object events
Change object events are generated when an object is changed. The event message contains the following event data: Change object.

Delete object events
Delete object events are generated when an object is deleted. The event message contains the following event data: Delete object.

Refresh object events
Refresh object events are generated by an explicit request to refresh. The event message contains the following event data: Refresh object.

Command events:

Command events are reported when an MQSC or PCF command runs successfully.

A command event message contains information about the origin, context, and content of a command. For example, a command event message is generated with such information if the MQSC command, ALTER QLOCAL, runs successfully.

The event messages for command events are put on the SYSTEM.ADMIN.COMMAND.EVENT queue.

Command events contain the following event data: Command.

Logger events:

Logger events are reported when a queue manager that uses linear logging starts writing log records to a new log extent IBM i or, on IBM i, to a new journal receiver. z/OS Logger events are not available with IBM MQ for z/OS.

A logger event message contains information specifying the log extents required by the queue manager to restart the queue manager, or for media recovery.

The event messages for logger events are put on the SYSTEM.ADMIN.LOGGER.EVENT queue.

The logger event message contains the following event data: Logger.

Event message data summary:

Use this summary to obtain information about the event data that each type of event message can contain.

Event type	**See these topics**
Authority events	Not Authorized (type 1)
	Not Authorized (type 2)
	Not Authorized (type 3)
	Not Authorized (type 4)
	Not Authorized (type 5)
	Not Authorized (type 6)
Channel events	Channel Activated
	Channel Auto-definition Error
	Channel Auto-definition OK
	Channel Blocked
	Channel Conversion Error
	Channel Not Activated
	Channel Started
	Channel Stopped
	Channel Stopped By User
Command events	Command
Configuration events	Create object
	Change object
	Delete object
	Refresh object
IMS bridge events	Bridge Started
	Bridge Stopped
Inhibit events	Get Inhibited
	Put Inhibited
Local events	Alias Base Queue Type Error
	Unknown Alias Base Queue
	Unknown Object Name
Logger events	Logger
Performance events	Queue Depth High
	Queue Depth Low
	Queue Full
	Queue Service Interval High
	Queue Service Interval OK

Event type	See these topics
Remote events	Default Transmission Queue Type Error
	Default Transmission Queue Usage Error
	Queue Type Error
	Remote Queue Name Error
	Transmission Queue Type Error
	Transmission Queue Usage Error
	Unknown Default Transmission Queue
	Unknown Remote Queue Manager
	Unknown Transmission Queue
SSL events	Channel SSL Error
Start and stop events	Queue Manager Active
	Queue Manager Not Active

Controlling events:

You enable and disable events by specifying the appropriate values for queue manager, queue attributes, or both, depending on the type of event.

You must enable each instrumentation event that you want to be generated. For example, the conditions causing a Queue Full event are:
- Queue Full events are enabled for a specified queue, and
- An application issues an MQPUT request to put a message on that queue, but the request fails because the queue is full.

Enable and disable events by using any of the following techniques:
- IBM MQ script commands (MQSC).
- The corresponding IBM MQ PCF commands.
- z/OS The operations and control panels for queue managers on z/OS.
- The IBM MQ Explorer.

Note: You can set attributes related to events for both queues and queue managers only by command. The MQI call MQSET does not support attributes related to events.

Related concepts:

"Instrumentation events" on page 930
An instrumentation event is a logical combination of conditions that a queue manager or channel instance detects and puts a special message, called an *event message*, on an event queue.

Related reference:

"Event types" on page 932
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information:

The MQSC commands

Automating administration tasks

Using Programmable Command Formats

Introducing the operations and control panels

Controlling queue manager events:

You control queue manager events by using queue manager attributes. To enable queue manager events, set the appropriate queue manager attribute to `ENABLED`. To disable queue manager events, set the appropriate queue manager attribute to `DISABLED`.

To enable or disable queue manager events, use the MQSC command ALTER QMGR, specifying the appropriate queue manager attribute. Table 101 summarizes how to enable queue manager events. To disable a queue manager event, set the appropriate parameter to `DISABLED`.

Table 101. Enabling queue manager events using MQSC commands

Event	**ALTER QMGR parameter**
Authority	AUTHOREV (ENABLED)
Inhibit	INHIBTEV (ENABLED)
Local	LOCALEV (ENABLED)
Remote	REMOTEEV (ENABLED)
Start and Stop	STRSTPEV (ENABLED)

Controlling channel and bridge events:

You control channel events by using queue manager attributes. To enable channel events, set the appropriate queue manager attribute to `ENABLED`. To disable channel events, set the appropriate queue manager attribute to `DISABLED`.

To enable or disable channels events use the MQSC command ALTER QMGR, specifying the appropriate queue manager attribute. Table 102 on page 941 summarizes how you enable channel and bridge events. To disable a queue manager event, set the appropriate parameter to `DISABLED`.

Restriction: z/OS Channel auto-definition events are not available on IBM MQ for z/OS.

Table 102. Enabling channel and bridge events using MQSC commands

Event	ALTER QMGR parameter
Channel	CHLEV (ENABLED)
Related to channel errors only	CHLEV (EXCEPTION)
IMS bridge	BRIDGEEV (ENABLED)
SSL	SSLEV (ENABLED)
Channel auto-definition	CHADEV(ENABLED)

With CHLEV set to exception, the following return codes, and corresponding reason qualifiers are generated:
- MQRC_CHANNEL_ACTIVATED
- MQRC_CHANNEL_CONV_ERROR
- MQRC_CHANNEL_NOT_ACTIVATED
- MQRC_CHANNEL_STOPPED
  - with the following ReasonQualifiers:
    - MQRQ_CHANNEL_STOPPED_ERROR
    - MQRQ_CHANNEL_STOPPED_RETRY
    - MQRQ_CHANNEL_STOPPED_DISABLED
- MQRC_CHANNEL_STOPPED_BY_USER
- MQRC_CHANNEL_BLOCKED
  - with the following ReasonQualifiers:
    - MQRQ_CHANNEL_BLOCKED_NOACCESS
    - MQRQ_CHANNEL_BLOCKED_USERID
    - MQRQ_CHANNEL_BLOCKED_ADDRESS

Controlling performance events:

You control performance events using the PERFMEV queue manager attribute. To enable performance events, set PERFMEV to `ENABLED`. To disable performance events, set the PERFMEV queue manager attribute to `DISABLED`.

To set the PERFMEV queue manager attribute to `ENABLED`, use the following MQSC command:

```
ALTER QMGR PERFMEV (ENABLED)
```

To enable specific performance events, set the appropriate queue attribute. Also, specify the conditions that cause the event.

Queue depth events

By default, all queue depth events are disabled. To configure a queue for any of the queue depth events:
1. Enable performance events on the queue manager.
2. Enable the event on the required queue.
3. Set the limits, if required, to the appropriate levels, expressed as a percentage of the maximum queue depth.

Queue service interval events

To configure a queue for queue service interval events you must:
1. Enable performance events on the queue manager.
2. Set the control attribute for a Queue Service Interval High or OK event on the queue as required.

3. Specify the service interval time by setting the QSVCINT attribute for the queue to the appropriate length of time.

Note: When enabled, a queue service interval event can be generated at any appropriate time, not necessarily waiting until an MQI call for the queue is issued. However, if an MQI call is used on a queue to put or remove a message, any applicable performance event is generated at that time. The event is *not* generated when the elapsed time becomes equal to the service interval time.

Controlling configuration, command, and logger events:

You control configuration, command, and logger events by using the queue manager attributes CONFIGEV, CMDEV, and LOGGEREV. To enable these events, set the appropriate queue manager attribute to ENABLED. To disable these events, set the appropriate queue manager attribute to DISABLED.

Configuration events
To enable configuration events, set CONFIGEV to ENABLED. To disable configuration events, set CONFIGEV to DISABLED. For example, you can enable configuration events by using the following MQSC command:

```
ALTER QMGR CONFIGEV (ENABLED)
```

Command events
To enable command events, set CMDEV to ENABLED. To enable command events for commands except DISPLAY MQSC commands and Inquire PCF commands, set the CMDEV to NODISPLAY. To disable command events, set CMDEV to DISABLED. For example, you can enable command events by using the following MQSC command:

```
ALTER QMGR CMDEV (ENABLED)
```

Logger events
To enable logger events, set LOGGEREV to ENABLED. To disable logger events, set LOGGEREV to DISABLED. For example, you can enable logger events by using the following MQSC command:

```
ALTER QMGR LOGGEREV(ENABLED)
```

Event queues:

When an event occurs, the queue manager puts an event message on the defined event queue. The event message contains information about the event.

You can define event queues, either as:
- Local queues
- Alias queues
- Local definitions of remote queues, or as
- Remote cluster queues

If you define all your event queues as local definitions of the same remote queue on one queue manager, you can centralize your monitoring activities.

You must not define event queues as transmission queues, because event messages have formats that are incompatible with the message format that is required for transmission queues.

Shared event queues are local queues defined with the QSGDISP(SHARED) value.

For more information about defining shared queues on z/OS, see Application programming with shared queues.

When an event queue is unavailable

If an event occurs when the event queue is not available, the event message is lost. For example, if you do not define an event queue for a category of event, all event messages for that category are lost. The event messages are not, for example, saved on the dead-letter (undelivered-message) queue.

However, you can define the event queue as a remote queue. Then, if there is a problem on the remote system putting messages to the resolved queue, the event message arrives on the dead-letter queue of the remote system.

An event queue might be unavailable for many different reasons including:
- The queue has not been defined.
- The queue has been deleted.
- The queue is full.
- The queue has been put-inhibited.

The absence of an event queue does not prevent the event from occurring. For example, after a performance event, the queue manager changes the queue attributes and resets the queue statistics. This change happens whether the event message is put on the performance event queue or not. The same is true in the case of configuration and command events.

Using triggered event queues

You can set up the event queues with triggers so that when an event is generated, the event message being put onto the event queue starts a user-written monitoring application. This application can process the event messages and take appropriate action. For example, certain events might require an operator to be informed, other events might start an application that performs some administration tasks automatically.

Event queues can have trigger actions associated with them and can create trigger messages. However, if these trigger messages in turn cause conditions that would normally generate an event, no event is generated. not generating an event in this instance ensures that looping does not occur.

Related concepts:

"Controlling events" on page 939
You enable and disable events by specifying the appropriate values for queue manager, queue attributes, or both, depending on the type of event.

"Format of event messages"
Event messages contain information about an event and its cause. Like other IBM MQ messages, an event message has two parts: a message descriptor and the message data.

Related information:

Application programming with shared queues

QSGDisp (MQLONG)

Conditions for a trigger event

Format of event messages:

Event messages contain information about an event and its cause. Like other IBM MQ messages, an event message has two parts: a message descriptor and the message data.
- The message descriptor is based on the MQMD structure.
- The message data consists of an *event header* and the *event data*. The event header contains the reason code that identifies the event type. Putting the event message, and any subsequent action, does not affect the reason code returned by the MQI call that caused the event. The event data provides further information about the event.

Typically, you process event messages with a system management application tailored to meet the requirements of the enterprise at which it runs.

When the queue managers in a queue sharing group detect the conditions for generating an event message, several queue managers can generate an event message for the shared queue, resulting in several event messages. To ensure that a system can correlate multiple event messages from different queue managers, these event messages have a unique correlation identifier *(CorrelId)* set in the message descriptor (MQMD).

Related reference:

"Activity report MQMD (message descriptor)" on page 1024
Use this page to view the values contained by the MQMD structure for an activity report

"Activity report MQEPH (Embedded PCF header)" on page 1028
Use this page to view the values contained by the MQEPH structure for an activity report

"Activity report MQCFH (PCF header)" on page 1030
Use this page to view the PCF values contained by the MQCFH structure for an activity report

Related information:

Event message reference

Event message format

Event message MQMD (message descriptor)

Event message MQCFH (PCF header)

Event message descriptions

Performance events

Performance events relate to conditions that can affect the performance of applications that use a specified queue. The scope of performance events is the queue. **MQPUT** calls and **MQGET** calls on one queue do not affect the generation of performance events on another queue.

Performance event messages can be generated at any appropriate time, not necessarily waiting until an MQI call for the queue is issued. However, if you use an MQI call on a queue to put or remove a message, any appropriate performance events are generated at that time.

Every performance event message that is generated is placed on the queue, SYSTEM.ADMIN.PERFM.EVENT.

The event data contains a reason code that identifies the cause of the event, a set of performance event statistics, and other data. The types of event data that can be returned in performance event messages are described in the following list:

- Queue Depth High
- Queue Depth Low
- Queue Full
- Queue Service Interval High
- Queue Service Interval OK

Examples that illustrate the use of performance events assume that you set queue attributes by using the appropriate IBM MQ commands (MQSC). On z/OS, you can also set queue attributes using the operations and controls panels for queue managers.

Related reference:

"Event types" on page 932
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Performance event statistics:

The performance event data in the event message contains statistics about the event. Use the statistics to analyze the behavior of a specified queue.

The event data in the event message contains information about the event for system management programs. For all performance events, the event data contains the names of the queue manager and the queue associated with the event. The event data also contains statistics related to the event. Table 103 summarizes the event statistics that you can use to analyze the behavior of a queue. All the statistics refer to what has happened since the last time the statistics were reset.

Table 103. Performance event statistics

Parameter	Description
TimeSinceReset	The elapsed time since the statistics were last reset.
HighQDepth	The maximum number of messages on the queue since the statistics were last reset.
MsgEnqCount	The number of messages enqueued (the number of MQPUT calls to the queue), since the statistics were last reset.
MsgDeqCount	The number of messages dequeued (the number of MQGET calls to the queue), since the statistics were last reset.

Performance event statistics are reset when any of the following changes occur:
- A performance event occurs (statistics are reset on all active queue managers).
- A queue manager stops and restarts.
- The PCF command, Reset Queue Statistics, is issued from an application program.
- z/OS On z/OS only, the RESET QSTATS command is issued at the console.

Related concepts:

"Performance events" on page 944
Performance events relate to conditions that can affect the performance of applications that use a specified queue. The scope of performance events is the queue. **MQPUT** calls and **MQGET** calls on one queue do not affect the generation of performance events on another queue.

"The service timer" on page 947
Queue service interval events use an internal timer, called the *service timer*, which is controlled by the queue manager. The service timer is used only if a queue service interval event is enabled.

"Rules for queue service interval events" on page 948
Formal rules control when the service timer is set and queue service interval events are generated.

Related tasks:

"Enabling queue service interval events" on page 948
To configure a queue for queue service interval events you set the appropriate queue manager and queue attributes.

Related information:

Queue Depth High

Reset Queue Statistics

RESET QSTATS

Queue service interval events:

Queue service interval events indicate whether an operation was performed on a queue within a user-defined time interval called the *service interval*. Depending on your installation, you can use queue service interval events to monitor whether messages are being taken off queues quickly enough.

Queue service interval events are *not* supported on shared queues.

The following types of queue service interval events can occur, where the term *get operation* refers to an **`MQGET`** call or an activity that removes a messages from a queue, such as using the **`CLEAR QLOCAL`** command:

Queue Service Interval OK

Indicates that after one of the following operations:

- An MQPUT call
- A get operation that leaves a non-empty queue

a get operation was performed within a user-defined time period, known as the *service interval*.

Only a get operation can cause the Queue Service Interval OK event message. Queue Service Interval OK events are sometimes described as OK events.

Queue Service Interval High

Indicates that after one of the following operations:

- An MQPUT call
- A get operation that leaves a non-empty queue

a get operation was **not** performed within a user-defined service interval.

Either a get operation or an MQPUT call can cause the Queue Service Interval High event message. Queue Service Interval High events are sometimes described as High events.

To enable both Queue Service Interval OK and Queue Service Interval High events, set the `QServiceIntervalEvent` control attribute to High. Queue Service Interval OK events are automatically enabled when a Queue Service Interval High event is generated. You do not need to enable Queue Service Interval OK events independently.

OK and High events are mutually exclusive, so if one is enabled the other is disabled. However, both events can be simultaneously disabled.

Figure 88 shows a graph of queue depth against time. At time P1, an application issues an MQPUT, to put a message on the queue. At time G1, another application issues an MQGET to remove the message from the queue.

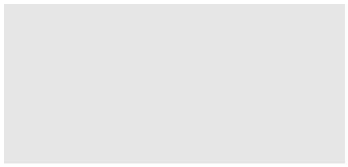


Figure 88. Understanding queue service interval events

The possible outcomes of queue service interval events are as follows:

- If the elapsed time between the put and the get is less than or equal to the service interval:
  - A *Queue Service Interval OK* event is generated at time G1, if queue service interval events are enabled
- If the elapsed time between the put and get is greater than the service interval:
  - A *Queue Service Interval High* event is generated at time G1, if queue service interval events are enabled.

The algorithm for starting the service timer and generating events is described in "Rules for queue service interval events" on page 948.

Related information:

Queue Service Interval OK

Queue Service Interval High

QServiceIntervalEvent (MQLONG)

QServiceIntervalEvent (10-digit signed integer)

ServiceIntervalEvent property

The service timer:

Queue service interval events use an internal timer, called the *service timer*, which is controlled by the queue manager. The service timer is used only if a queue service interval event is enabled.

What precisely does the service timer measure?
The service timer measures the elapsed time between an MQPUT call to an empty queue or a get operation, and the next put or get, provided the queue depth is nonzero between these two operations.

When is the service timer active?
The service timer is always active (running), if the queue has messages on it (depth is nonzero) and a queue service interval event is enabled. If the queue becomes empty (queue depth zero), the timer is put into an OFF state, to be restarted on the next put.

When is the service timer reset?
The service timer is always reset after a get operation . It is also reset by an MQPUT call to an empty queue. However, it is not necessarily reset on a queue service interval event.

How is the service timer used?
Following a get operation or an MQPUT call, the queue manager compares the elapsed time as measured by the service timer, with the user-defined service interval. The result of this comparison is that:

- An OK event is generated if there is a get operation and the elapsed time is less than or equal to the service interval, AND this event is enabled.
- A high event is generated if the elapsed time is greater than the service interval, AND this event is enabled.

Can applications read the service timer?
No, the service timer is an internal timer that is not available to applications.

What about the *TimeSinceReset* parameter?
The *TimeSinceReset* parameter is returned as part of the event statistics in the event data. It specifies the time between successive queue service interval events, unless the event statistics are reset.

Rules for queue service interval events:

Formal rules control when the service timer is set and queue service interval events are generated.

Rules for the service timer

The service timer is reset to zero and restarted as follows:
- After an MQPUT call to an empty queue.
- After an MQGET call, if the queue is not empty after the MQGET call.

The resetting of the timer does not depend on whether an event has been generated.

At queue manager startup the service timer is set to startup time if the queue depth is greater than zero.

If the queue is empty following a get operation, the timer is put into an OFF state.

Queue Service Interval High events

The Queue Service Interval event must be enabled (set to `HIGH`).

Queue Service Interval High events are automatically enabled when a Queue Service Interval OK event is generated.

If the service time is greater than the service interval, an event is generated on, or before, the next MQPUT or get operation.

Queue Service Interval OK events

Queue Service Interval OK events are automatically enabled when a Queue Service Interval High event is generated.

If the service time (elapsed time) is less than or equal to the service interval, an event is generated on, or before, the next get operation.
Related tasks:
"Enabling queue service interval events"
To configure a queue for queue service interval events you set the appropriate queue manager and queue attributes.

Enabling queue service interval events:

To configure a queue for queue service interval events you set the appropriate queue manager and queue attributes.

About this task

The high and OK events are mutually exclusive; that is, when one is enabled, the other is automatically disabled:
- When a high event is generated on a queue, the queue manager automatically disables high events and enables OK events for that queue.
- When an OK event is generated on a queue, the queue manager automatically disables OK events and enables high events for that queue.

Table 104. Enabling queue service interval events using MQSC

Queue service interval event	Queue attributes
Queue Service Interval High Queue Service Interval OK No queue service interval events	QSVCIEV (HIGH) QSVCIEV (OK) QSVCIEV (NONE)
Service interval	QSVCINT (*tt*) where *tt* is the service interval time in milliseconds.

Perform the following steps to enable queue service interval events:

Procedure

1. Set the queue manager attribute PERFMEV to ENABLED. Performance events are enabled on the queue manager.
2. Set the control attribute, QSVCIEV, for a Queue Service Interval High or OK event on the queue, as required.
3. Set the QSVCINT attribute for the queue to specify the appropriate service interval time.

Example

To enable Queue Service Interval High events with a service interval time of 10 seconds (10 000 milliseconds) use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)

ALTER QLOCAL('MYQUEUE') QSVCINT(10000) QSVCIEV(HIGH)
```

.

Queue service interval events examples:

Use the examples in this section to understand the information that you can obtain from queue service interval events.

The three subtopic examples provide progressively more complex illustrations of the use of queue service interval events.

The figures accompanying the examples in each subtopic have the same structure:

- Figure 1 is a graph of queue depth against time, showing individual MQGET calls and MQPUT calls.
- The Commentary section shows a comparison of the time constraints. There are three time periods that you must consider:
  - The user-defined service interval.
  - The time measured by the service timer.
  - The time since event statistics were last reset (TimeSinceReset in the event data).
- The Event statistics summary section shows which events are enabled at any instant and what events are generated.

The examples illustrate the following aspects of queue service interval events:

- How the queue depth varies over time.
- How the elapsed time as measured by the service timer compares with the service interval.
- Which event is enabled.
- Which events are generated.

Remember: Example 1 shows a simple case where the messages are intermittent and each message is removed from the queue before the next one arrives. From the event data, you know that the maximum number of messages on the queue was one. You can, therefore, work out how long each message was on the queue.

However, in the general case, where there is more than one message on the queue and the sequence of MQGET calls and MQPUT calls is not predictable, you cannot use queue service interval events to calculate how long an individual message remains on a queue. The TimeSinceReset parameter, which is returned in the event data, can include a proportion of time when there are no messages on the queue. Therefore any results you derive from these statistics are implicitly averaged to include these times.

Related concepts:

"Queue service interval events" on page 946

Queue service interval events indicate whether an operation was performed on a queue within a user-defined time interval called the *service interval*. Depending on your installation, you can use queue service interval events to monitor whether messages are being taken off queues quickly enough.

"The service timer" on page 947

Queue service interval events use an internal timer, called the *service timer*, which is controlled by the queue manager. The service timer is used only if a queue service interval event is enabled.

Queue service interval events: example 1:

A basic sequence of MQGET calls and MQPUT calls, where the queue depth is always one or zero.

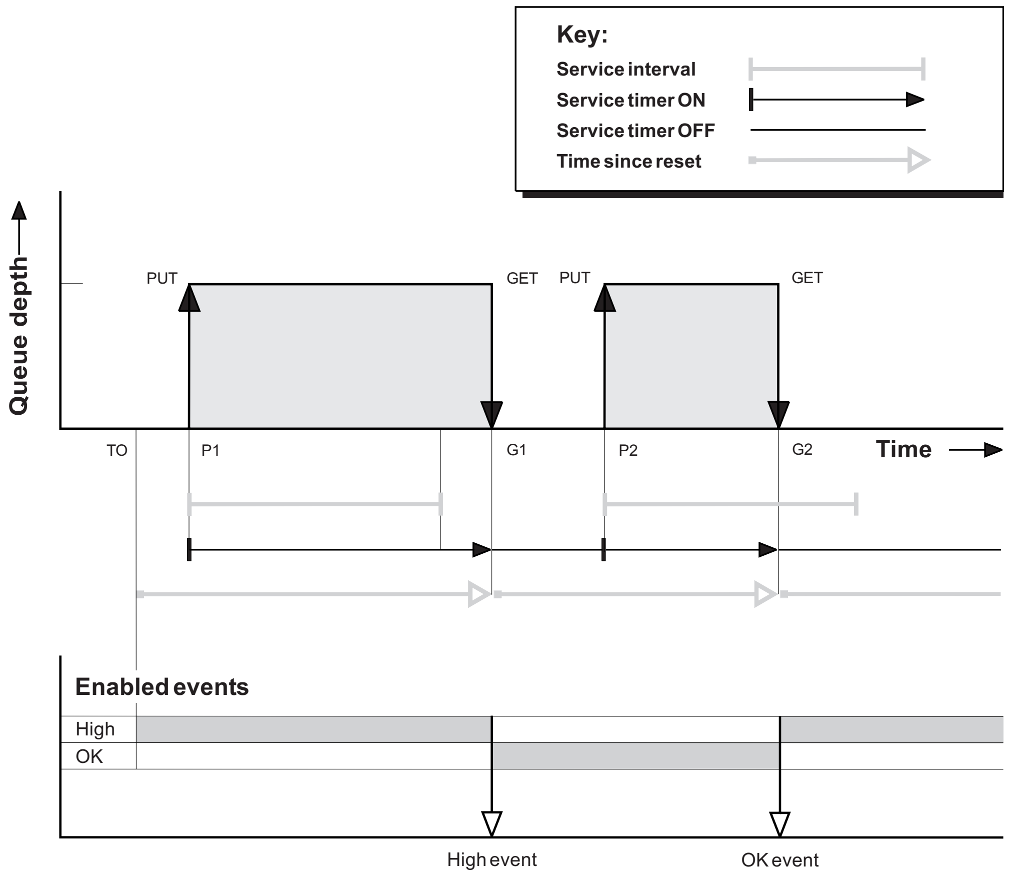


Figure 89. Queue service interval events - example 1

Commentary

1. At P1, an application puts a message onto an empty queue. This starts the service timer.

   Note that T0 might be queue manager startup time.
2. At G1, another application gets the message from the queue. Because the elapsed time between P1 and G1 is greater than the service interval, a Queue Service Interval High event is generated on the MQGET call at G1. When the high event is generated, the queue manager resets the event control attribute so that:
   a. The OK event is automatically enabled.
   b. The high event is disabled.

   Because the queue is now empty, the service timer is switched to an OFF state.
3. At P2, a second message is put onto the queue. This restarts the service timer.
4. At G2, the message is removed from the queue. However, because the elapsed time between P2 and G2 is less than the service interval, a Queue Service Interval OK event is generated on the MQGET call at G2. When the OK event is generated, the queue manager resets the control attribute so that:
   a. The high event is automatically enabled.
   b. The OK event is disabled.

   Because the queue is empty, the service timer is again switched to an OFF state.

Event statistics summary

Table 105 summarizes the event statistics for this example.

Table 105. Event statistics summary for example 1

	Event 1	**Event 2**
Time of event	T(G1)	T(G2)
Type of event	High	OK
TimeSinceReset	T(G1) - T(0)	T(G2) - T(G1)
HighQDepth	1	1
MsgEnqCount	1	1
MsgDeqCount	1	1

The middle part of Figure 89 on page 950 shows the elapsed time as measured by the service timer compared to the service interval for that queue. To see whether a queue service interval event might occur, compare the length of the horizontal line representing the service timer (with arrow) to that of the line representing the service interval. If the service timer line is longer, and the Queue Service Interval High event is enabled, a Queue Service Interval High event occurs on the next get. If the timer line is shorter, and the Queue Service Interval OK event is enabled, a Queue Service Interval OK event occurs on the next get.

Queue service interval events: example 2:

A sequence of MQPUT calls and MQGET calls, where the queue depth is not always one or zero.

This example also shows instances of the timer being reset without events being generated, for example, at time P2.

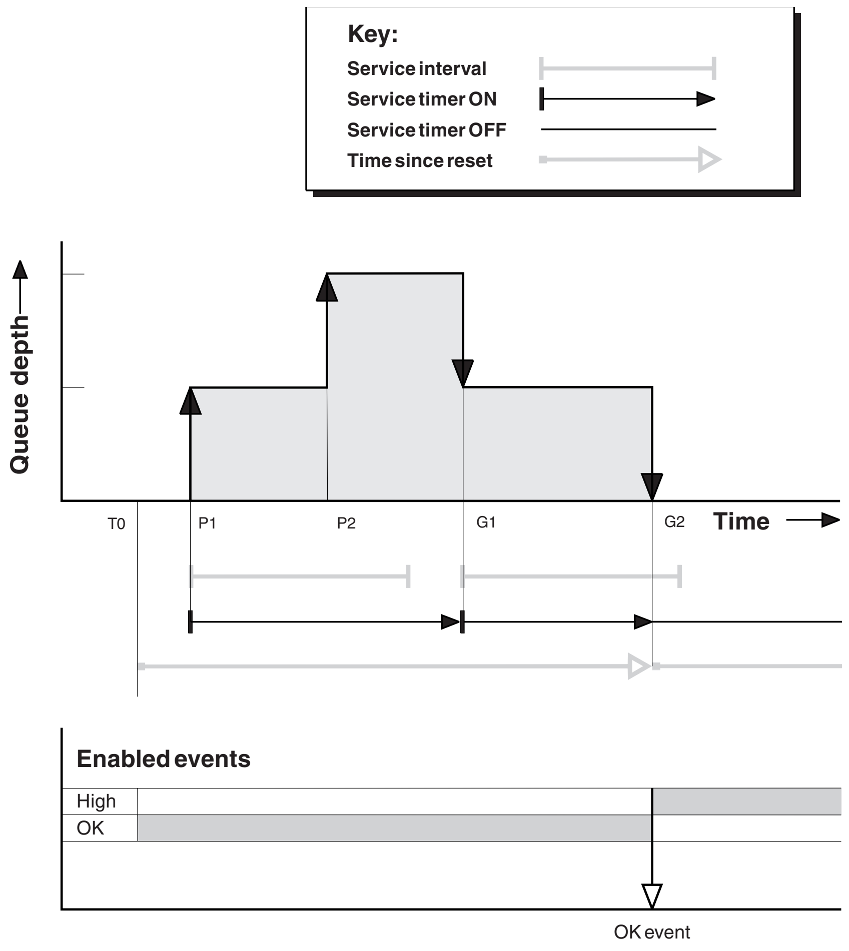


Figure 90. Queue service interval events - example 2

Commentary

In this example, OK events are enabled initially and queue statistics were reset at time T0.

1. At P1, the first put starts the service timer.
2. At P2, the second put does not generate an event because a put cannot cause an OK event.
3. At G1, the service interval has now been exceeded and therefore an OK event is not generated. However, the MQGET call causes the service timer to be reset.
4. At G2, the second get occurs within the service interval and this time an OK event is generated. The queue manager resets the event control attribute so that:
   a. The high event is automatically enabled.
   b. The OK event is disabled.

   Because the queue is now empty, the service timer is switched to an OFF state.

Event statistics summary

Table 106 summarizes the event statistics for this example.

Table 106. Event statistics summary for example 2

	Event 2
Time of event	T(G2)
Type of event	OK
TimeSinceReset	T(G2) - T(0)
HighQDepth	2
MsgEnqCount	2
MsgDeqCount	2

Queue service interval events: example 3:

A sequence of MQGET calls and MQPUT calls that is more sporadic than the previous examples.

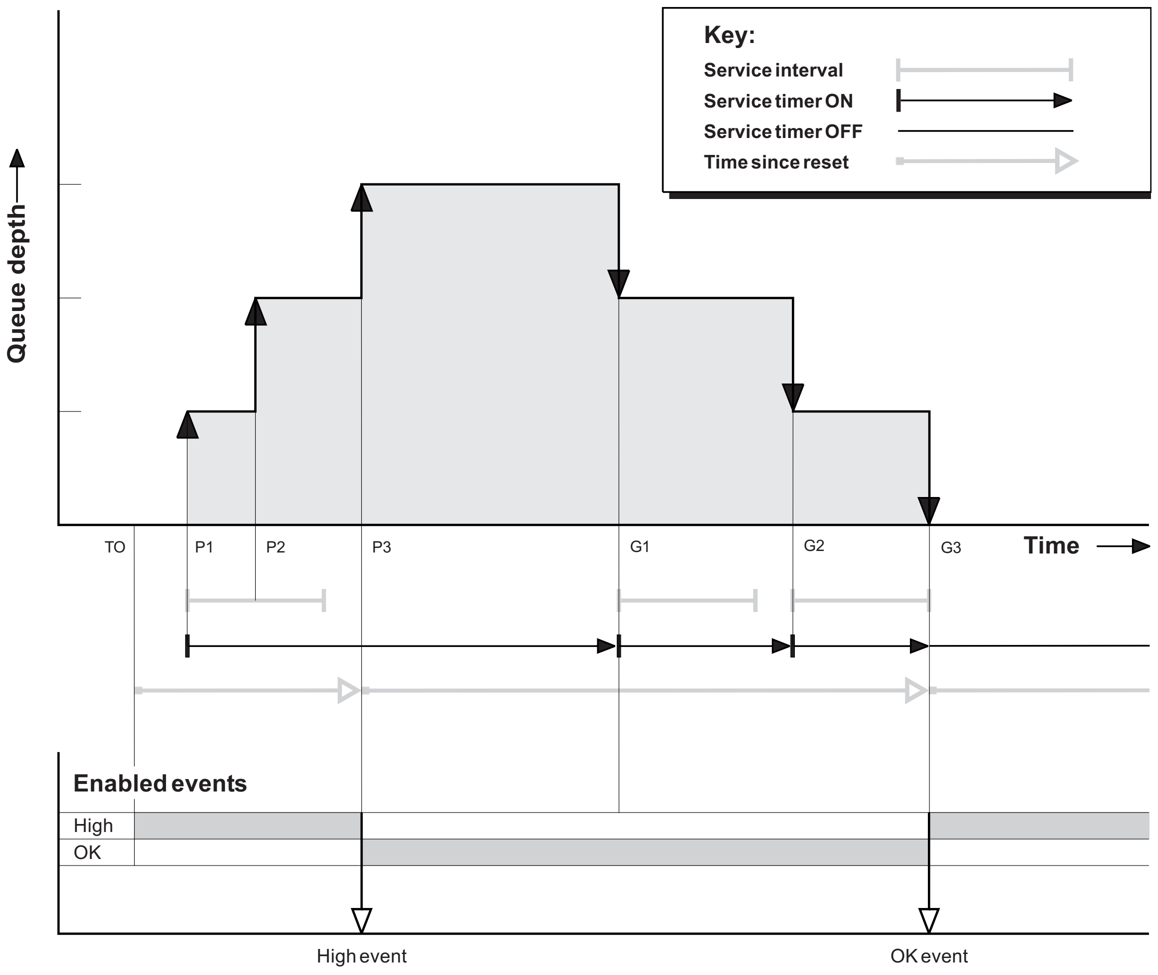


Figure 91. Queue service interval events - example 3

Commentary

1. At time T(0), the queue statistics are reset and Queue Service Interval High events are enabled.
2. At P1, the first put starts the service timer.
3. At P2, the second put increases the queue depth to two. A high event is not generated here because the service interval time has not been exceeded.
4. At P3, the third put causes a high event to be generated. (The timer has exceeded the service interval.) The timer is not reset because the queue depth was not zero before the put. However, OK events are enabled.
5. At G1, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. The MQGET call does, however, reset the service timer.
6. At G2, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. Again, the MQGET call resets the service timer.
7. At G3, the third get empties the queue and the service timer is *equal* to the service interval. Therefore an OK event is generated. The service timer is reset and high events are enabled. The MQGET call empties the queue, and this puts the timer in the OFF state.

Event statistics summary

Table 107 summarizes the event statistics for this example.

Table 107. Event statistics summary for example 3

	Event 1	**Event 2**
Time of event	T(P3)	T(G3)
Type of event	High	OK
TimeSinceReset	T(P3) - T(0)	T(G3) - T(P3)
HighQDepth	3	3
MsgEnqCount	3	0
MsgDeqCount	0	3

Queue depth events:

Queue depth events are related to the queue depth, that is, the number of messages on the queue.

In IBM MQ applications, queues must not become full. If they do, applications can no longer put messages on the queue that they specify. Although the message is not lost if this occurs, a full queue can cause considerable inconvenience. The number of messages can build up on a queue if the messages are being put onto the queue faster than the applications that process them can take them off.

The solution to this problem depends on the particular circumstances, but might involve:

- Diverting some messages to another queue.
- Starting new applications to take more messages off the queue.
- Stopping nonessential message traffic.
- Increasing the queue depth to overcome a transient maximum.

Advance warning that problems might be on their way makes it easier to take preventive action. For this purpose, IBM MQ provides the following queue depth events:

Queue Depth High events
: Indicate that the queue depth has increased to a predefined threshold called the Queue Depth High limit.

Queue Depth Low events
Indicate that the queue depth has decreased to a predefined threshold called the Queue Depth Low limit.

Queue Full events
Indicate that the queue has reached its maximum depth, that is, the queue is full.

A Queue Full Event is generated when an application attempts to put a message on a queue that has reached its maximum depth. Queue Depth High events give advance warning that a queue is filling up. This means that having received this event, the system administrator needs to take some preventive action. You can configure the queue manager such that, if the preventive action is successful and the queue depth drops to a safer level, the queue manager generates a Queue Depth Low event.

The first queue depth event example illustrates the effect of presumed action preventing the queue becoming full.

Related concepts:

"Queue depth events examples" on page 958
Use these examples to understand the information that you can obtain from queue depth events

Related information:

Queue Full

Queue Depth High

Queue Depth Low

Enabling queue depth events:

To configure a queue for any of the queue depth events you set the appropriate queue manager and queue attributes.

About this task

By default, all queue depth events are disabled. When enabled, queue depth events are generated as follows:

- A Queue Depth High event is generated when a message is put on the queue, causing the queue depth to be greater than or equal to the value determined by the Queue Depth High limit.
  - A Queue Depth High event is automatically enabled by a Queue Depth Low event on the same queue.
  - A Queue Depth High event automatically enables both a Queue Depth Low and a Queue Full event on the same queue.
- A Queue Depth Low event is generated when a message is removed from a queue by a get operation causing the queue depth to be less than or equal to the value determined by the Queue Depth Low limit.
  - A Queue Depth Low event is automatically enabled by a Queue Depth High event or a Queue Full event on the same queue.
  - A Queue Depth Low event automatically enables both a Queue Depth High and a Queue Full event on the same queue.
- A Queue Full event is generated when an application is unable to put a message onto a queue because the queue is full.
  - A Queue Full event is automatically enabled by a Queue Depth High or a Queue Depth Low event on the same queue.
  - A Queue Full event automatically enables a Queue Depth Low event on the same queue.

Perform the following steps to configure a queue for any of the queue depth events:

Procedure

1. Enable performance events on the queue manager, using the queue manager attribute PERFMEV.
2. Set one of the following attributes to enable the event on the required queue:
   - *QDepthHighEvent* (QDPHIEV in MQSC)
   - *QDepthLowEvent* (QDPLOEV in MQSC)
   - *QDepthMaxEvent* (QDPMAXEV in MQSC)
3. Optional: To set the limits, assign the following attributes, as a percentage of the maximum queue depth:
   - *QDepthHighLimit* (QDEPTHHI in MQSC)
   - *QDepthLowLimit* (QDEPTHLO in MQSC)

   Restriction: QDEPTHHI must not be less than QDEPTHLO.

   If QDEPTHHI equals QDEPTHLO an event message is generated every time the queue depth passes the value in either direction, because the high threshold is enabled when the queue depth is below the value and the low threshold is enabled when the depth is above the value.

Results

Note:

A Queue Depth Low event is not generated when expired messages are removed from a queue by a get operation causing the queue depth to be less than, or equal to, the value determined by the Queue Depth Low limit.

IBM MQ generates the low event message only during a successful get operation. Therefore, when the expired messages are removed from the queue, no queue depth low event message is generated.

Additionally, after the removal of these expired messages from the queue, queue depth high event and queue depth low event are not reset.

Example

To enable Queue Depth High events on the queue MYQUEUE with a limit set at 80%, use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDEPTHHI(80) QDPHIEV(ENABLED)
```

To enable Queue Depth Low events on the queue MYQUEUE with a limit set at 20%, use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDEPTHLO(20) QDPLOEV(ENABLED)
```

To enable Queue Full events on the queue MYQUEUE, use the following MQSC commands:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDPMAXEV(ENABLED)
```

Shared queues and queue depth events (IBM MQ for z/OS):

Event monitoring is more straightforward for an application that uses shared queues if all the queue managers in the queue-sharing group have the same setting for the PERFMEV attribute.

When a queue depth event occurs on a shared queue, and the queue manager attribute PERFMEV is set to ENABLED, the queue managers in the queue-sharing group produce an event message. If PERFMEV is set to DISABLED on some of the queue managers, event messages are not produced by those queue managers, making event monitoring from an application more difficult. For more straightforward monitoring, give each queue manager the same setting for the PERFMEV attribute.

This event message that each queue manager generates represents its individual usage of the shared queue. If a queue manager performs no activity on the shared queue, various values in the event message are null or zero. You can use null event messages as follows:
- Ensure that each active queue manager in a queue-sharing group generates one event message
- Highlight cases of no activity on a shared queue for the queue manager that produced the event message

Coordinating queue manager

When a queue manager issues a queue depth event, it updates the shared queue object definition to toggle the active performance event attributes. For example, depending on the definition of the queue attributes, a Queue Depth High event enables a Queue Depth Low and a Queue Full event. After updating the shared queue object successfully, the queue manager that detected the performance event initially becomes the *coordinating queue manager*.

If enabled for performance events, the coordinating queue manager performs the following actions:
1. Issues an event message that captures all shared queue performance data it has gathered since the last time an event message was created, or since the queue statistics were last reset. The message descriptor (MQMD) of this message contains a unique correlation identifier (*CorrelId*) created by the coordinating queue manager.
2. Broadcasts to all other *active* queue managers in the same queue-sharing group to request the production of an event message for the shared queue. The broadcast contains the correlation identifier created by the coordinating queue manager for the set of event messages.

Having received a request from the coordinating queue manager, if there is an active queue manager in the queue-sharing group that is enabled for performance events , that active queue manager issues an event message for the shared queue. The event message that is issued contains information about all the operations performed by the receiving (active) queue manager since the last time an event message was created, or since the statistics were last reset. The message descriptor (MQMD) of this event message contains the unique correlation identifier (*CorrelId*) specified by the coordinating queue manager.

When performance events occur on a shared queue, *n* event messages are produced, where *n* is a number from 1 to the number of active queue managers in the queue-sharing group. Each event message contains data that relates to the shared queue activity for the queue manager that generated the event message.

You can view event message data for a shared queue using the following views:

Queue-sharing view
Collects all data from event messages with the same correlation identifier.

Queue manager view

Each event message shows how much it has been used by its originating queue manager.

Differences between shared and nonshared queues

Enabling queue depth events on shared queues differs from enabling them on nonshared queues. A key difference is that events are switched on for shared queues even if PERFMEV is DISABLED on the queue manager. This is not the case for nonshared queues.

Consider the following example, which illustrates this difference:

- QM1 is a queue manager with `PerformanceEvent` (PERFMEV in MQSC) set to DISABLED.
- SQ1 is a shared queue with QSGDISP set to (SHARED) QLOCAL in MQSC.
- LQ1 is a nonshared queue with QSGDISP set to (QMGR) QLOCAL in MQSC.

Both queues have the following attributes set on their definitions:

- QDPHIEV (ENABLED)
- QDPLOEV (DISABLED)
- QDPMAXEV (DISABLED)

If messages are placed on both queues so that the depth meets or exceeds the QDEPTHHI threshold, the QDPHIEV value on SQ1 switches to DISABLED. Also, QDPLOEV and QDPMAXEV are switched to ENABLED. SQ1's attributes are automatically switched for each performance event at the time the event criteria are met.

In contrast the attributes for LQ1 remain unchanged until PERFMEV on the queue manager is ENABLED. This means that if the queue manager's PERFMEV attribute is ENABLED, DISABLED and then re-ENABLED for example, the performance event settings on shared queues might not be consistent with those of nonshared queues, even though they might have initially been the same.

Queue depth events examples:

Use these examples to understand the information that you can obtain from queue depth events

The first example provides a basic illustration of queue depth events. The second example is more extensive, but the principles are the same as for the first example. Both examples use the same queue definition, as follows:

The queue, MYQUEUE1, has a maximum depth of 1000 messages. The high queue depth limit is 80% and the low queue depth limit is 20%. Initially, Queue Depth High events are enabled, while the other queue depth events are disabled.

The IBM MQ commands (MQSC) to configure this queue are:

```
ALTER QMGR PERFMEV(ENABLED)

DEFINE QLOCAL('MYQUEUE1') MAXDEPTH(1000) QDPMAXEV(DISABLED) QDEPTHHI(80)
QDPHIEV(ENABLED) QDEPTHLO(20) QDPLOEV(DISABLED)
```

Related concepts:

"Queue depth events" on page 954
Queue depth events are related to the queue depth, that is, the number of messages on the queue.

Related tasks:

"Enabling queue depth events" on page 955
To configure a queue for any of the queue depth events you set the appropriate queue manager and queue attributes.

Related information:

The MQSC commands

Queue depth events: example 1:

A basic sequence of queue depth events.

Figure 92 shows the variation of queue depth over time.

Figure 92. Queue depth events (1)

Commentary

1. At T(1), the queue depth is increasing (more MQPUT calls than MQGET calls) and crosses the Queue Depth Low limit. No event is generated at this time.
2. The queue depth continues to increase until T(2), when the depth high limit (80%) is reached and a Queue Depth High event is generated.

   This enables both Queue Full and Queue Depth Low events.
3. The (presumed) preventive actions instigated by the event prevent the queue from becoming full. By time T(3), the Queue Depth High limit has been reached again, this time from above. No event is generated at this time.

4. The queue depth continues to fall until T(4), when it reaches the depth low limit (20%) and a Queue Depth Low event is generated.

   This enables both Queue Full and Queue Depth High events.

Event statistics summary

Table 108 summarizes the queue event statistics and Table 109 summarizes which events are enabled.

Table 108. Event statistics summary for queue depth events (example 1)

	Event 2	**Event 4**
Time of event	T(2)	T(4)
Type of event	Queue Depth High	Queue Depth Low
TimeSinceReset	T(2) - T(0)	T(4) - T(2)
HighQDepth (Maximum queue depth since reset)	800	900
MsgEnqCount	1157	1220
MsgDeqCount	357	1820

Table 109. Summary showing which events are enabled

Time period	**Queue Depth High event**	**Queue Depth Low event**	**Queue Full event**
Before T(1)	ENABLED	-	-
T(1) to T(2)	ENABLED	-	-
T(2) to T(3)	-	ENABLED	ENABLED
T(3) to T(4)	-	ENABLED	ENABLED
After T(4)	ENABLED	-	ENABLED

Queue depth events: example 2:

A more extensive sequence of queue depth events.

Figure 93 on page 961 shows the variation of queue depth over time.

Figure 93. Queue depth events (2)

Commentary

1. No Queue Depth Low event is generated at the following times:
   - T(1) (Queue depth increasing, and not enabled)
   - T(2) (Not enabled)
   - T(3) (Queue depth increasing, and not enabled)
2. At T(4) a Queue Depth High event occurs. This enables both Queue Full and Queue Depth Low events.
3. At T(9) a Queue Full event occurs **after** the first message that cannot be put on the queue because the queue is full.
4. At T(12) a Queue Depth Low event occurs.

Event statistics summary

Table 110 on page 962 summarizes the queue event statistics and Table 111 on page 962 summarizes which events are enabled at different times for this example.

Table 110. Event statistics summary for queue depth events (example 2)

	Event 4	Event 6	Event 8	Event 9	Event 12
Time of event	T(4)	T(6)	T(8)	T(9)	T(12)
Type of event	Queue Depth High	Queue Depth Low	Queue Depth High	Queue Full	Queue Depth Low
TimeSinceReset	T(4) - T(0)	T(6) - T(4)	T(8) - T(6)	T(9) - T(8)	T(12) - T(9)
HighQDepth	800	855	800	1000	1000
MsgEnqCount	1645	311	1377	324	221
MsgDeqCount	845	911	777	124	1021

Table 111. Summary showing which events are enabled

Time period	Queue Depth High event	Queue Depth Low event	Queue Full event
T(0) to T(4)	ENABLED	-	-
T(4) to T(6)	-	ENABLED	ENABLED
T(6) to T(8)	ENABLED	-	ENABLED
T(8) to T(9)	-	ENABLED	ENABLED
T(9) to T(12)	-	ENABLED	-
After T(12)	ENABLED	-	ENABLED

Note: Events are out of syncpoint. Therefore you could have an empty queue, then fill it up causing an event, then roll back all of the messages under the control of a syncpoint manager. However, event enabling has been automatically set, so that the next time the queue fills up, no event is generated.

Configuration events

Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Configuration events notify you about changes to the attributes of an object. There are four types of configuration events:

- Create object events
- Change object events
- Delete object events
- Refresh object events

The event data contains the following information:

Origin information
: comprises the queue manager from where the change was made, the ID of the user that made the change, and how the change came about, for example by a console command.

Context information
: a replica of the context information in the message data from the command message.

: Context information is included in the event data only when the command was entered as a message on the SYSTEM.COMMAND.INPUT queue.

Object identity
: comprises the name, type and disposition of the object.

Object attributes
: comprises the values of all the attributes in the object.

In the case of change object events, two messages are generated, one with the information before the change, the other with the information after.

Every configuration event message that is generated is placed on the queue SYSTEM.ADMIN.CONFIG.EVENT.

Related concepts:

"Configuration events" on page 937
Configuration events are generated when a configuration event is requested explicitly, or automatically when an object is created, modified, or deleted.

Related reference:

"Event types" on page 932
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information:

Create object

Change object

Delete object

Refresh object

Configuration event generation:

Use this page to view the commands that cause configuration events to be generated and to understand the circumstances in which configuration events are not generated

A configuration event message is put to the configuration event queue when the CONFIGEV queue manager attribute is ENABLED and

- any of the following commands, or their PCF equivalent, are issued:
  - DELETE AUTHINFO
  - DELETE CFSTRUCT
  - DELETE CHANNEL
  - DELETE NAMELIST
  - DELETE PROCESS
  - DELETE QMODEL/QALIAS/QREMOTE
  - DELETE STGCLASS
  - DELETE TOPIC
  - REFRESH QMGR
- any of the following commands, or their PCF equivalent, are issued even if there is no change to the object:
  - DEFINE/ALTER AUTHINFO
  - DEFINE/ALTER CFSTRUCT
  - DEFINE/ALTER CHANNEL
  - DEFINE/ALTER NAMELIST
  - DEFINE/ALTER PROCESS
  - DEFINE/ALTER QMODEL/QALIAS/QREMOTE
  - DEFINE/ALTER STGCLASS
  - DEFINE/ALTER TOPIC
  - DEFINE MAXSMSGS
  - SET CHLAUTH

- ALTER QMGR, unless the CONFIGEV attribute is DISABLED and is not changed to ENABLED

- any of the following commands, or their PCF equivalent, are issued for a local queue that is not temporary dynamic, even if there is no change to the queue.
  - DELETE QLOCAL
  - DEFINE/ALTER QLOCAL
- an MQSET call is issued, other than for a temporary dynamic queue, even if there is no change to the object.

When configuration events are not generated

Configuration events messages are not generated in the following circumstances:

- When a command or an MQSET call fails
- When a queue manager encounters an error trying to put a configuration event on the event queue, in which case the command or MQSET call completes, but no event message is generated
- For a temporary dynamic queue
- When internal changes are made to the TRIGGER queue attribute
- For the configuration event queue SYSTEM.ADMIN.CONFIG.EVENT, except by the REFRESH QMGR command
- For REFRESH/RESET CLUSTER and RESUME/SUSPEND QMGR commands that cause clustering changes
- When Creating or deleting a queue manager

Related concepts:

"Configuration events" on page 962
Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Related information:

The MQSC commands

Introduction to Programmable Command Formats

MQSET - Set object attributes

MQSET - Set object attributes

Configuration event usage:

Use this page to view how you can use configuration events to obtain information about your system, and to understand the factors, such as CMDSCOPE, that can affect your use of configuration events.

You can use configuration events for the following purposes:

1. To produce and maintain a central configuration repository, from which reports can be produced and information about the structure of the system can be generated.
2. To generate an audit trail. For example, if an object is changed unexpectedly, information regarding who made the alteration and when it was done can be stored.

   This can be particularly useful when command events are also enabled. If an MQSC or PCF command causes a configuration event and a command event to be generated, both event messages will share the same correlation identifier in their message descriptor.

For an MQSET call or any of the following commands:

- DEFINE object
- ALTER object
- DELETE object

if the queue manager attribute CONFIGEV is enabled, but the configuration event message cannot be put on the configuration event queue, for example the event queue has not been defined, the command or MQSET call is executed regardless.

Effects of CMDSCOPE

For commands where CMDSCOPE is used, the configuration event message or messages will be generated on the queue manager or queue managers where the command is executed, not where the command is entered. However, all the origin and context information in the event data will relate to the original command as entered, even where the command using CMDSCOPE is one that has been generated by the source queue manager.

Where a queue sharing group includes queue managers that are not at the current version, events will be generated for any command that is executed by means of CMDSCOPE on a queue manager that is at the current version, but not on those that are at a previous version. This happens even if the queue manager where the command is entered is at the previous version, although in such a case no context information is included in the event data.

Related concepts:

"Configuration events" on page 962
Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Related information:

Introduction to Programmable Command Formats

MQSET - Set object attributes

MQSET - Set object attributes

Refresh Object configuration event:

The Refresh Object configuration event is different from the other configuration events, because it occurs only when explicitly requested.

The create, change, and delete events are generated by an MQSET call or by a command to change an object but the refresh object event occurs only when explicitly requested by the MQSC command, REFRESH QMGR, or its PCF equivalent.

The REFRESH QMGR command is different from all the other commands that generate configuration events. All the other commands apply to a particular object and generate a single configuration event for that object. The REFRESH QMGR command can produce many configuration event messages potentially representing every object definition stored by a queue manager. One event message is generated for each object that is selected.

The REFRESH QMGR command uses a combination of three selection criteria to filter the number of objects involved:

- Object Name
- Object Type
- Refresh Interval

If you specify none of the selection criteria on the REFRESH QMGR command, the default values are used for each selection criteria and a refresh configuration event message is generated for every object definition stored by the queue manager. This might cause unacceptable processing times and event message generation. Consider specifying some selection criteria.

The REFRESH QMGR command that generates the refresh events can be used in the following situations:

- When configuration data is wanted about all or some of the objects in a system regardless of whether the objects have been recently manipulated, for example, when configuration events are first enabled.

  Consider using several commands, each with a different selection of objects, but such that all are included.
- If there has been an error in the SYSTEM.ADMIN.CONFIG.EVENT queue. In this circumstance, no configuration event messages are generated for Create, Change, or Delete events. When the error on the queue has been corrected, the Refresh Queue Manager command can be used to request the generation of event messages, which were lost while there was an error in the queue. In this situation consider setting the refresh interval to the time for which the queue was unavailable.

Related concepts:

"Configuration events" on page 962
Configuration events are notifications that are generated when an object is created, changed, or deleted, and can also be generated by explicit requests.

Related information:

REFRESH QMGR

Refresh Queue Manager

Command events

Command events are notifications that an MQSC, or PCF command has run successfully.

The event data contains the following information:

Origin information
comprises the queue manager from where the command was issued, the ID of the user that issued the command, and how the command was issued, for example by a console command.

Context information
a replica of the context information in the message data from the command message. If a command is not entered using a message, context information is omitted.

Context information is included in the event data only when the command was entered as a message on the SYSTEM.COMMAND.INPUT queue.

Command information
the type of command that was issued.

Command data
- for PCF commands, a replica of the command data
- for MQSC commands, the command text

The command data format does not necessarily match the format of the original command. For example, on distributed platforms the command data format is always in PCF format, even if the original request was an MQSC command.

Every command event message that is generated is placed on the command event queue, SYSTEM.ADMIN.COMMAND.EVENT.

Related reference:

"Event types" on page 932
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information:

Command

Command event generation:

Use this page to view the situations that cause command events to be generated and to understand the circumstances in which command events are not generated

A command event message is generated in the following situations:
- When the CMDEV queue manager attribute is specified as ENABLED and an MQSC or PCF command runs successfully.
- When the CMDEV queue manager attribute is specified as NODISPLAY and any command runs successfully, with the exception of DISPLAY commands (MQSC), and Inquire commands (PCF).
- When you run the MQSC command, ALTER QMGR, or the PCF command, Change Queue Manager, and the CMDEV queue manager attribute meets either of the following conditions:
  - CMDEV is not specified as DISABLED after the change
  - CMDEV was not specified as DISABLED before the change

If a command runs against the command event queue, SYSTEM.ADMIN.COMMAND.EVENT, a command event is generated if the queue still exists and it is not put-inhibited.

When command events are not generated

A command event message is not generated in the following circumstances:
- When a command fails
- When a queue manager encounters an error trying to put a command event on the event queue, in which case the command runs regardless, but no event message is generated
- For the MQSC command REFRESH QMGR TYPE (EARLY)
- For the MQSC command START QMGR MQSC
- For the MQSC command SUSPEND QMGR, if the parameter LOG is specified
- For the MQSC command RESUME QMGR, if the parameter LOG is specified

Related concepts:

"Command events" on page 966
Command events are notifications that an MQSC, or PCF command has run successfully.

Related information:

REFRESH QMGR

START QMGR

SUSPEND QMGR

RESUME QMGR

SUSPEND QMGR, RESUME QMGR and clusters

Command event usage:

Use this page to view how you can use command events to generate an audit trail of the commands that have run

For example, if an object is changed unexpectedly, information regarding who made the alteration and when it was done can be stored. This can be particularly useful when configuration events are also enabled. If an MQSC or PCF command causes a command event and a configuration event to be generated, both event messages will share the same correlation identifier in their message descriptor.

If a command event message is generated, but cannot be put on the command event queue, for example if the command event queue has not been defined, the command for which the command event was generated still runs regardless.

Effects of CMDSCOPE

For commands where CMDSCOPE is used, the command event message or messages will be generated on the queue manager or queue managers where the command runs, not where the command is entered. However, all the origin and context information in the event data will relate to the original command as entered, even where the command using CMDSCOPE is one that has been generated by the source queue manager.

Related concepts:

"Command events" on page 966
Command events are notifications that an MQSC, or PCF command has run successfully.

"Command event generation" on page 967
Use this page to view the situations that cause command events to be generated and to understand the circumstances in which command events are not generated

Related information:

The MQSC commands

PCF commands and responses in groups

Logger events

Logger events are notifications that a queue manager has started writing to a new log extent IBM i or, on IBM i, a journal receiver. z/OS Logger event messages are not available with IBM MQ for z/OS.

The event data contains the following information:

- The name of the current log extent.
- The name of the earliest log extent needed for restart recovery.
- The name of the earliest log extent needed for media recovery.
- The directory in which the log extents are located.

Every logger event message that is generated is placed on the logger event queue, SYSTEM.ADMIN.LOGGER.EVENT.

Related reference:

"Event types" on page 932
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Related information:

Logger

Logger event generation:

Use this page to view the situations that cause logger events to be generated and to understand the circumstances in which logger events are not generated

A logger event message is generated in the following situations:

- When the LOGGEREV queue manager attribute is specified as ENABLED and the queue manager starts writing to a new log extent or, on IBM i, a journal receiver.
- When the LOGGEREV queue manager attribute is specified as ENABLED and the queue manager starts.
- When the LOGGEREV queue manager attribute is changed from DISABLED to ENABLED.

Tip: You can use the RESET QMGR MQSC command to request a queue manager to start writing to a new log extent.

When logger events are not generated

A logger event message is not generated in the following circumstances:

- When a queue manager is configured to use circular logging.

  In this case, the LOGGEREV queue manager attribute is set as DISABLED and cannot be altered.
- When a queue manager encounters an error trying to put a logger event on the event queue, in which case the action that caused the event completes, but no event message is generated.

Related concepts:

"Logger events" on page 968
Logger events are notifications that a queue manager has started writing to a new log extent IBM i or, on IBM i, a journal receiver. z/OS Logger event messages are not available with IBM MQ for z/OS.

Related information:

LoggerEvent (MQLONG)

LoggerEvent (10-digit signed integer)

RESET QMGR

Logger event usage:

Use this page to view how you can use logger events to determine the log extents that are no longer required for queue manager restart, or media recovery.

You can archive superfluous log extents to a medium such as tape for disaster recovery before removing them from the active log directory. Regular removal of superfluous log extents keeps disk space usage to a minimum.

If the LOGGEREV queue manager attribute is enabled, but a logger event message cannot be put on the logger event queue, for example because the event queue has not been defined, the action that caused the event continues regardless.

Related concepts:

"Logger events" on page 968
Logger events are notifications that a queue manager has started writing to a new log extent
IBM i or, on IBM i, a journal receiver. z/OS Logger event messages are not available with IBM MQ for z/OS.

Related reference:

"Logger event generation" on page 969
Use this page to view the situations that cause logger events to be generated and to understand the circumstances in which logger events are not generated

Related information:

LoggerEvent (MQLONG)

LoggerEvent (10-digit signed integer)

Sample program to monitor the logger event queue:

Use this page to view a sample C program that monitors the logger event queue for new event messages, reads those messages, and puts the contents of the message to stdout.

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSLOG0.C                                                   */
/*                                                                            */
/* Description:  Sample C program to monitor the logger event queue and output*/
/*               a message to stdout when a logger event occurs              */
/* <N_OCO_COPYRIGHT>                                                          */
/* Licensed Materials - Property of IBM                                       */
/*                                                                            */
/* 63H9336                                                                    */
/* (c) Copyright IBM Corp. 2005 All Rights Reserved.                          */
/*                                                                            */
/* US Government Users Restricted Rights - Use, duplication or                */
/* disclosure restricted by GSA ADP Schedule Contract with                    */
/* IBM Corp.                                                                  */
/* <NOC_COPYRIGHT>                                                            */
/******************************************************************************/
/*                                                                            */
/* Function: AMQSLOG is a sample program which monitors the logger event      */
/* queue for new event messages, reads those messages, and puts the contents  */
/* of the message to stdout.                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSLOG has 1 parameter - the queue manager name (optional, if not         */
/* specified then the default queue manager is implied)                       */
/*                                                                            */
/******************************************************************************/

/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <cmqc.h>       /* MQI constants*/
#include <cmqcfc.h>     /* PCF constants*/

/******************************************************************************/
/* Constants                                                                  */
/******************************************************************************/
```

```
#define   MAX_MESSAGE_LENGTH   8000

typedef struct _ParmTableEntry
{
  MQLONG  ConstVal;
  PMQCHAR Desc;
} ParmTableEntry;

ParmTableEntry ParmTable[] =
{
  0                             ,"",
  MQCA_Q_MGR_NAME               ,"Queue Manager Name",
  MQCMD_LOGGER_EVENT            ,"Logger Event Command",
  MQRC_LOGGER_STATUS            ,"Logger Status",
  MQCACF_CURRENT_LOG_EXTENT_NAME,"Current Log Extent",
  MQCACF_RESTART_LOG_EXTENT_NAME,"Restart Log Extent",
  MQCACF_MEDIA_LOG_EXTENT_NAME  ,"Media Log Extent",
  MQCACF_LOG_PATH               ,"Log Path"};

/*******************************************************************************/
/* Function prototypes                                                         */
/*******************************************************************************/

static void ProcessPCF(MQHCONN    hConn,
                       MQHOBJ     hEventQueue,
                       PMQCHAR    pBuffer);

static PMQCHAR ParmToString(MQLONG Parameter);

/***********************************************************************/
/* Function: main                                                      */
/***********************************************************************/
int main(int argc, char * argv[])
{
  MQLONG    CompCode;
  MQLONG    Reason;
  MQHCONN   hConn = MQHC_UNUSABLE_HCONN;
  MQOD      ObjDesc = { MQOD_DEFAULT };
  MQCHAR    QMName[MQ_Q_MGR_NAME_LENGTH+1] = "";
  MQCHAR    LogEvQ[MQ_Q_NAME_LENGTH] = "SYSTEM.ADMIN.LOGGER.EVENT";
  MQHOBJ    hEventQueue;
  PMQCHAR   pBuffer = NULL;

  printf("\n/*************************************/\n");
  printf("/* Sample Logger Event Monitor start */\n");
  printf("/*************************************/\n");

  /*******************************************************************/
  /* Parse any command line options                                  */
  /*******************************************************************/

  if (argc > 1)
     strncpy(QMName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);

  pBuffer = (char *)malloc(MAX_MESSAGE_LENGTH);
  if (!pBuffer)
  {
    printf("Can't allocate %d bytes\n",MAX_MESSAGE_LENGTH);
    goto MOD_EXIT;
  }

  /*******************************************************************/
  /* Connect to the specified (or default) queue manager             */
  /*******************************************************************/

  MQCONN(QMName,
         &hConn,
         &CompCode,
         &Reason);

  if (Reason != MQCC_OK)
  {
    printf("Error in call to MQCONN, Reason %d, CompCode %d\n", Reason,
    CompCode);
    goto MOD_EXIT;
  }

  /* Open the logger event queue for input  */

  strncpy(ObjDesc.ObjectQMgrName,QMName, MQ_Q_MGR_NAME_LENGTH);
  strncpy(ObjDesc.ObjectName, LogEvQ, MQ_Q_NAME_LENGTH);

  MQOPEN(  hConn,
          &ObjDesc,
           MQOO_INPUT_EXCLUSIVE,
          &hEventQueue,
          &CompCode,
          &Reason);
  if (Reason)
  {
    printf("MQOPEN failed for queue manager %.48s Queue %.48s Reason: %d\n",
                               ObjDesc.ObjectQMgrName,
                                   ObjDesc.ObjectName,
                                   Reason);
    goto MOD_EXIT;
  }
  else
  {
    ProcessPCF(hConn, hEventQueue, pBuffer);
  }

  MOD_EXIT:

  if (pBuffer != NULL) {
    free(pBuffer);
  }

  /*******************************************************************/
  /* Disconnect                                                      */
  /*******************************************************************/
  if (hConn != MQHC_UNUSABLE_HCONN) {
     MQDISC(&hConn, &CompCode, &Reason);
  }

  return 0;
}

/***************************************************************************/
/* Function: ProcessPCF                                                    */
/***************************************************************************/
/*                                                                         */
/* Input Parameters:  Handle to queue manager connection                   */
/*                    Handle to the opened logger event queue object       */
/*                    Pointer to a memory buffer to store the incoming PCF msg*/
/*                                                                         */
/* Output Parameters: None                                                 */
/*                                                                         */
/* Logic: Wait for messages to appear on the logger event queue and display */
/* their contents.                                                         */
/*                                                                         */
/***************************************************************************/

static void ProcessPCF(MQHCONN    hConn,
                       MQHOBJ     hEventQueue,
                      PMQCHAR    pBuffer)
{
  MQCFH   * pCfh;
  MQCFST  * pCfst;
  MQGMO     Gmo     = { MQGMO_DEFAULT };
  MQMD      Mqmd    = { MQMD_DEFAULT };
  PMQCHAR   pPCFCmd;
  MQLONG    Reason  = 0;
  MQLONG    CompCode;
  MQLONG    MsgLen;
  PMQCHAR   Parm = NULL;
                                          /* Set timeout value           */
  Gmo.Options      |= MQGMO_WAIT;
  Gmo.Options |= MQGMO_CONVERT;
  Gmo.WaitInterval = MQWI_UNLIMITED;
  /**********************************************************************/
  /* Process response Queue                                             */
  /**********************************************************************/
  while (Reason == MQCC_OK)
  {
    memcpy(&Mqmd.MsgId;   , MQMI_NONE, sizeof(Mqmd.MsgId));
    memset(&Mqmd.CorrelId, 0, sizeof(Mqmd.CorrelId));

    MQGET( hConn,
           hEventQueue,
          &Mqmd,
          &Gmo,
           MAX_MESSAGE_LENGTH,
           pBuffer,
          &MsgLen,
          &CompCode,
          &Reason);
    if (Reason != MQCC_OK)
    {
      switch(Reason)
      {
        case MQRC_NO_MSG_AVAILABLE:
             printf("Timed out");
             break;

        default:
             printf("MQGET failed RC(%d)\n", Reason);
             break;
      }
      goto MOD_EXIT;
    }

    /********************************************************************/
    /* Only expect PCF event messages on this queue                     */
    /********************************************************************/
    if (memcmp(Mqmd.Format, MQFMT_EVENT, sizeof(Mqmd.Format)))
    {
     printf("Unexpected message format '%8.8s' received\n",Mqmd.Format);
     continue;
    }


    /*********************************************************************/
    /* Build the output by parsing the received PCF message, first the */
    /* header, then each of the parameters                              */
    /*********************************************************************/

    pCfh = (MQCFH *)pBuffer;

    if (pCfh -> Reason)
    {
     printf("-----------------------------------------------------------------\n");
```

```
      printf("Event Message Received\n");

      Parm = ParmToString(pCfh->Command);
      if (Parm != NULL) {
        printf("Command  :%s \n",Parm);
      }
      else
      {
        printf("Command  :%d \n",pCfh->Command);
      }

      printf("CompCode :%d\n"   ,pCfh->CompCode);

      Parm = ParmToString(pCfh->Reason);
      if (Parm != NULL) {
        printf("Reason   :%s \n",Parm);
      }
      else
      {
        printf("Reason   :%d \n",pCfh->Reason);
      }
     }

     pPCFCmd  = (char *)  (pCfh+1);
     printf("----------------------------------------------------------------\n");
     while(pCfh -> ParameterCount--)
     {
       pCfst = (MQCFST *) pPCFCmd;
       switch(pCfst -> Type)
       {
         case MQCFT_STRING:
              Parm = ParmToString(pCfst -> Parameter);
              if (Parm != NULL) {
                printf("%-32s",Parm);
              }
              else
              {
                printf("%-32d",pCfst -> Parameter);
              }

              fwrite( pCfst -> String, pCfst -> StringLength, 1, stdout);
              pPCFCmd += pCfst -> StrucLength;
              break;
         default:
              printf("Unrecoginised datatype %d returned\n",pCfst->Type);
              goto MOD_EXIT;
       }
       putchar('\n');
     }
     printf("----------------------------------------------------------------\n");
   }
MOD_EXIT:

 return;
}

/******************************************************************************/
/* Function: ParmToString                                                     */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Parameter for which to get string description          */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Takes a parameter as input and returns a pointer to a string       */
/* description for that parameter, or NULL if the parameter does not  */
/* have an associated string description                                     */
```

```
/******************************************************************************/

static PMQCHAR ParmToString(MQLONG Parameter){
  long i;
  for (i=0 ; i< sizeof(ParmTable)/sizeof(ParmTableEntry); i++)
  {
    if (ParmTable[i].ConstVal == Parameter ParmTable[i].Desc)
      return ParmTable[i].Desc;
  }
  return NULL;
}
```

Sample output

This application produces the following form of output:

```
/************************************/
/* Sample Logger Event Monitor start */
/************************************/
-----------------------------------------------------------------
Event Message Received
Command  :Logger Event Command
CompCode :0
Reason   :Logger Status
-----------------------------------------------------------------
Queue Manager Name              CSIM

Current Log Extent              AMQA000001
Restart Log Extent              AMQA000001
Media Log Extent                AMQA000001
Log Path                        QMCSIM
-----------------------------------------------------------------
```

Related concepts:

"Logger event usage" on page 969
Use this page to view how you can use logger events to determine the log extents that are no longer required for queue manager restart, or media recovery.

"Command event usage" on page 968
Use this page to view how you can use command events to generate an audit trail of the commands that have run

Related reference:

"Logger event generation" on page 969
Use this page to view the situations that cause logger events to be generated and to understand the circumstances in which logger events are not generated

Authority configuration events

V 8.0.0.4

Authority configuration events are output when a change is made from any of the security control operations through the command line, MQSC, PCF, or corresponding iSeries commands.

The event data contains the following information:

Origin information
comprises the queue manager from where the change was made, the ID of the user that made the change, and how the change came about, for example by a console command.

Context information
a replica of the context information in the message data from the command message.

Context information is included in the event data when the command was entered as a message on the SYSTEM.ADMIN.COMMAND.QUEUE queue.

Authority Record identity
comprises the profile name, and object type of the authority record.

Object attributes
comprises the values of all the attributes in the authority record.

In the case of change authority record events, two messages are generated, one with the information before the change, the other with the information after the change.

Every event message that is generated is placed on the SYSTEM.ADMIN.CONFIG.EVENT queue.

Related reference:
"Event types" on page 932
Use this page to view the types of instrumentation event that a queue manager or channel instance can report

Authority configuration event generation: V 8.0.0.4

Use this page to view the situations that cause authority configuration events to be generated, and to understand the circumstances in which authority configuration events are not generated.

Authority configuration events notify you about changes to the attributes of an authority record. There are three types of authority configuration event:
- Change Authority Record
- Delete Authority Record
- Refresh Authority Record

An authority event message is put to the configuration event queue, when the **`CONFIGEV`** queue manager attribute is set to *ENABLED* and any of the following commands, or their MQSC equivalent, are issued, even if there is no actual change to the authority record:
- Delete Authority Record PCF command
- Set Authority Record PCF command
- setmqaut control command
- RVKMQMAUT CL command
- GRTMQMAUT CL command

When authority configuration events are not generated

The authority configuration event messages are not generated in the following circumstances:
- When a command fails
- When a queue manager encounters an error trying to put a message on the event queue, in which case the command completes, but no event message is generated
- When creating or deleting a queue manager
- When an object is deleted regardless of the **`AUTHREC`** option on the delete command. The corresponding command event shows that operation, which does not apply to the authority record for individual users.

Related concepts:

"Command events" on page 966
Command events are notifications that an MQSC, or PCF command has run successfully.

Related information:

REFRESH QMGR

Sample program to monitor instrumentation events

V 8.0.0.4 **`amqsevt`** formats the instrumentation events that a queue manager can create, and is supplied with IBM MQ. The program reads messages from event queues, and formats them into readable strings.

V 8.0.0.4

As a sample program, both source and binary are provided. The sample is provided on all the distributed platforms, including IBM i.

The single binary file `amqsevt` (or `amqsevt.exe`) is shipped in the samples fileset and is installed in the samples bin (`tools\c\samples\bin` or `bin64`) directory.

The source files `amqsevta.c` is also shipped in the samples fileset, and is installed in the samples directory, that is, `tools\c\samples` on Windows.

Note that the program can read from multiple event queues, and subscribe to multiple topics, by using MQCB to retrieve the messages.

When running as a client, the sample can connect to any queue manager including z/OS.

Attention: You can use the program without specifying any parameters, in which case the program attempts to connect to the default queue manager and read messages from the standard set of event queues (SYSTEM.ADMIN.*.EVENT).

In this situation, the program waits forever for messages, until you press the Enter key to end the program.

However, you are more likely to use the program with the various options described.

V 8.0.0.4

Syntax

```
►►─amqsevt─┬───────────────┬─┬──────────────────────────────────────────────────────────────┬──►
           └─ -m─QMgrName─┘ │                            (1)                               │
                             └─ -r─Reconnection Options───────┬─d─Reconnect Disabled───────┘
                                                              ├─r─Reconnect───────────────┤
                                                              └─m─Reconnect Queue manager─┘

►─┬───────────────────────┬─┬────────────────────────┬──────────────────────────────────────────►
  └─ -b─Browse Messages─┘ └─ -c─Connect as Client─┘

►─┬────────────────────────────────────────┬─┬───────────────┬─┬────────────────┬──────────────►
  └─ -d─Print Definitions without formatting─┘ └─ -u─User ID─┘ └─ -w─Wait Time─┘
```

```
                     (2)                 (2)
►─┬──────────────┬──┬──────────────┬──────────────────────────────────────────────►◄
  └─ -t──Topic───┘  └─ -q──Queue───┘
```

Notes:

1 Available only when being used as a client

2 Queues and topics can have multiple entries

V 8.0.0.4

Optional parameters

-m *QueueManagerName*
Specify a specific queue manager for reading events.

-r *Reconnection Options*
Auto reconnection options when used as a client. The possible values are:

d Reconnect the client disabled

r Reconnect the client

m Reconnect the queue manager

-b Browse records only, rather than destructively reading the messages

-c Selects connection as a client.

-d Selects the printing mode used in the second example. The MQI constants are printed exactly as they appear in the header files.

-u *User ID*
Specify a specific user and causes a prompt to appear requesting a password

-w *Wait*
Causes the program to exit if no event messages have arrived within the number of seconds specified.

Note that, if you do not specify a time, the program only ends normally when you press the Enter key.

-t *Topic* **and**
-q *Queue*
Both the **–q** and **–t** options can be given multiple times on the command line.

Therefore, it is possible to read from some standard queues and also from topics (if events are being sent to them) from a single run of the program.

If no queues or topics are named on the command line, the default event queues are opened.

Note: The program detects if it has connected to a z/OS queue manager as a client, and changes the default set of event queues appropriately, as z/OS does not have the SYSTEM.ADMIN.LOGGER.EVENT queue.

When topics are used, the program uses a non-durable subscription with a managed queue so that everything gets cleaned out when it exits.

V 8.0.0.4

Sample output

The following two examples show the output from the program.

The first example uses the default formatting option where the program takes the MQI definition of a field and formats the output to make the output more readable.

```
**** Message (320 Bytes) on Queue SYSTEM.ADMIN.QMGR.EVENT ****
Event Type                       : Queue Mgr Event
Reason                           : Unknown Alias Base Queue
Event created                    : 2015/06/17 13:47:07.02 GMT
  Queue Mgr Name                 : V8003_A
  Queue Name                     : EVT.NO.BASE.QUEUE
  Base Object Name               : EVT.NOT.DEFINED
  Appl Type                      : Unix
  Appl Name                      : amqsput
  Base Type                      : Queue
```

The second example shows the alternative formatting, using the -d option, that does not try to translate MQI constants. This might be preferable for some scripting tools that look for specific MQI values.

```
**** Message (320 Bytes) on Queue SYSTEM.ADMIN.QMGR.EVENT ****
Event Type                       : MQCMD_Q_MGR_EVENT
Reason                           : MQRC_UNKNOWN_ALIAS_BASE_Q
Event created                    : 2015/06/17 13:52:48.18 GMT
  MQCA_Q_MGR_NAME                : V8003_A
  MQCA_Q_NAME                    : EVT.NO.BASE.QUEUE
  MQCA_BASE_OBJECT_NAME          : EVT.NOT.DEFINED
  MQIA_APPL_TYPE                 : MQAT_UNIX
  MQCACF_APPL_NAME               : amqsput
  MQIA_BASE_TYPE                 : MQOT_Q
```

V 8.0.0.4

Example usage

The following example shows you how to use more than one queue:

```
amqsevt -m QM1 -q SYSTEM.ADMIN.QMGR.EVENT -q SYSTEM.ADMIN.PERM.EVENT -w 1
```

Related concepts:

"Event monitoring" on page 929
Event monitoring is the process of detecting occurrences of *instrumentation events* in a queue manager network. An instrumentation event is a logical combination of events that is detected by a queue manager or channel instance. Such an event causes the queue manager or channel instance to put a special message, called an *event message*, on an event queue.

"Instrumentation events" on page 930
An instrumentation event is a logical combination of conditions that a queue manager or channel instance detects and puts a special message, called an *event message*, on an event queue.

Related reference:

"Sample program to monitor the logger event queue" on page 970
Use this page to view a sample C program that monitors the logger event queue for new event messages, reads those messages, and puts the contents of the message to stdout.

Related information:

C programming

Message monitoring

Message monitoring is the process of identifying the route a message has taken through a queue manager network. By identifying the types of activities, and the sequence of activities performed on behalf of a message, the message route can be determined.

As a message passes through a queue manager network, various processes perform activities on behalf of the message. Use one of the following techniques to determine a message route:
- The IBM MQ display route application (dspmqrte)
- Activity recording
- Trace-route messaging

These techniques all generate special messages that contain information about the activities performed on the message as it passed through a queue manager network. Use the information returned in these special messages to achieve the following objectives:
- Record message activity.
- Determine the last known location of a message.
- Detect routing problems in your queue manager network.
- Assist in determining the causes of routing problems in your queue manager network.
- Confirm that your queue manager network is running correctly.
- Familiarize yourself with the running of your queue manager network.
- Trace published messages.

Related information:

Types of message

Activities and operations

Activities are discrete actions that an application performs on behalf of a message. Activities consist of operations, which are single pieces of work that an application performs.

The following actions are examples of activities:
- A message channel agent (MCA) sends a message from a transmission queue down a channel
- An MCA receives a message from a channel and puts it on its target queue
- An application getting a message from a queue, and putting a reply message in response.
- The IBM MQ publish/subscribe engine processes a message.

Activities consist of one or more *operations*. Operations are single pieces of work that an application performs. For example, the activity of an MCA sending a message from a transmission queue down a channel consists of the following operations:
1. Getting a message from a transmission queue (a *Get* operation).
2. Sending the message down a channel (a *Send* operation).

In a publish/subscribe network, the activity of the IBM MQ publish/subscribe engine processing a message can consist of the following multiple operations:
1. Putting a message to a topic string (a *Put* operation).
2. Zero or more operations for each of the subscribers that are considered for receipt of the message (a *Publish* operation, a *Discarded Publish* operation or an *Excluded Publish* operation).

Information from activities

You can identify the sequence of activities performed on a message by recording information as the message is routed through a queue manager network. You can determine the route of a message through the queue manager network from the sequence of activities performed on the message, and can obtain the following information:

The last known location of a message
: If a message does not reach its intended destination, you can determine the last known location of the message from a complete or partial message route.

Configuration issues with a queue manager network

When studying the route of a message through a queue manager network, you might see that the message has not gone where expected. There are many reasons why this can occur, for example, if a channel is inactive, the message might take an alternative route.

For a publish/subscribe application, you can also determine the route of a message being published to a topic and any messages that flow in a queue manager network as a result of being published to subscribers. In such situations, a system administrator can determine whether there are any problems in the queue manager network, and if appropriate, correct them.

Message routes

Depending on your reason for determining a message route, you can use the following general approaches:

Using activity information recorded for a trace-route message

Trace-route messages record activity information for a specific purpose. You can use them to determine configuration issues with a queue manager network, or to determine the last known location of a message. If a trace-route message is generated to determine the last known location of a message that did not reach its intended destination, it can mimic the original message. This gives the trace-route message the greatest chance of following the route taken by the original message.

The IBM MQ display route application can generate trace-route messages.

Using activity information recorded for the original message

You can enable any message for activity recording and have activity information recorded on its behalf. If a message does not reach its intended destination, you can use the recorded activity information to determine the last known location of the message. By using activity information from the original message, the most accurate possible message route can be determined, leading to the last known location. To use this approach, the original message must be enabled for activity recording.

Warning: Avoid enabling all messages in a queue manager network for activity recording. Messages enabled for activity recording can have many activity reports generated on their behalf. If every message in a queue manager network is enabled for activity recording, the queue manager network traffic can increase to an unacceptable level.

Related concepts:

"Message monitoring" on page 979
Message monitoring is the process of identifying the route a message has taken through a queue manager network. By identifying the types of activities, and the sequence of activities performed on behalf of a message, the message route can be determined.

"Message route techniques"
Activity recording and trace-route messaging are techniques that allow you to record activity information for a message as it is routed through a queue manager network.

"Trace-route messaging" on page 988
Trace-route messaging is a technique that uses *trace-route messages* to record activity information for a message. Trace-route messaging involves sending a trace-route message into a queue manager network.

Related information:

Writing your own message channel agents

Message route techniques

Activity recording and trace-route messaging are techniques that allow you to record activity information for a message as it is routed through a queue manager network.

Activity recording

If a message has the appropriate report option specified, it requests that applications generate *activity reports* as it is routed through a queue manager network. When an application performs an activity on behalf of a message, an activity report can be generated, and delivered to an appropriate location. An activity report contains information about the activity that was performed on the message.

The activity information collected using activity reports must be arranged in order before a message route can be determined.

Trace-route messaging

Trace-route messaging is a technique that involves sending a *trace-route message* into a queue manager network. When an application performs an activity on behalf of the trace-route message, activity information can be accumulated in the message data of the trace-route message, or activity reports can be generated. If activity information is accumulated in the message data of the trace-route message, when it reaches its target queue a trace-route reply message containing all the information from the trace-route message can be generated and delivered to an appropriate location.

Because a trace-route message is dedicated to recording the sequence of activities performed on its behalf, there are more processing options available compared with normal messages that request activity reports.

Comparison of activity recording and trace-route messaging

Both activity recording and trace-route messaging can provide activity information to determine the route a message has taken through a queue manager network. Both methods have their own advantages.

Benefit	**Activity recording**	**Trace-route messaging**
Can determine the last known location of a message	Yes	Yes
Can determine configuration issues with a queue manager network	Yes	Yes
Can be requested by any message (is not restricted to use with trace-route messages)	Yes	No
Message data is left unmodified	Yes	No
Message processed normally	Yes	No
Activity information can be accumulated in the message data	No	Yes
Optional message delivery to target queue	No	Yes
If a message is caught in an infinite loop, it can be detected and dealt with	No	Yes
Activity information can be put in order reliably	No	Yes
Application provided to display the activity information	No	Yes

Message route completeness

In some cases it is not possible to identify the full sequence of activities performed on behalf of a message, so only a partial message route can be determined. The completeness of a message route is directly influenced by the queue manager network that the messages are routed through. The completeness of a message route depends on the level of the queue managers in the queue manager network, as follows:

Queue managers at IBM WebSphere MQ Version 6.0 and subsequent releases

MCAs and user-written applications connected to queue managers at IBM WebSphere MQ Version 6.0 or subsequent releases can record information related to the activities performed on behalf of a message. The recording of activity information is controlled by the queue manager attributes ACTIVREC and ROUTEREC. If a queue manager network consists of queue managers at IBM WebSphere MQ Version 6.0 or subsequent releases only, complete message routes can be determined.

IBM MQ queue managers before Version 6.0
Applications connected to IBM MQ queue managers before Version 6.0 **do not** record the activities that they have performed on behalf of a message. If a queue manager network contains any IBM MQ queue manager prior to Version 6.0, only a partial message route can be determined.

How activity information is stored

IBM MQ stores activity information in activity reports, trace-route messages, or trace-route reply messages. In each case the information is stored in a structure called the *Activity* PCF group. A trace-route message or trace-route reply message can contain many Activity PCF groups, depending on the number of activities performed on the message. Activity reports contain one Activity PCF group because a separate activity report is generated for every recorded activity.

With trace-route messaging, additional information can be recorded. This additional information is stored in a structure called the *TraceRoute* PCF group. The TraceRoute PCF group contains a number of PCF structures that are used to store additional activity information, and to specify options that determine how the trace-route message is handled as it is routed through a queue manager network.

Related concepts:

"Activity recording"
Activity recording is a technique for determining the routes that messages take through a queue manager network. To determine the route that a message has taken, the activities performed on behalf of the message are recorded.

"Trace-route messaging" on page 988
Trace-route messaging is a technique that uses *trace-route messages* to record activity information for a message. Trace-route messaging involves sending a trace-route message into a queue manager network.

Related reference:

"The TraceRoute PCF group" on page 994
Attributes in the *TraceRoute* PCF group control the behavior of a trace-route message. The *TraceRoute* PCF group is in the message data of every trace-route message.

"Activity report message data" on page 1031
Use this page to view the parameters contained by the *Activity* PCF group in an activity report message. Some parameters are returned only when specific operations have been performed.

Activity recording

Activity recording is a technique for determining the routes that messages take through a queue manager network. To determine the route that a message has taken, the activities performed on behalf of the message are recorded.

When using activity recording, each activity performed on behalf of a message can be recorded in an activity report. An activity report is a type of report message. Each activity report contains information about the application that performed the activity on behalf of the message, when the activity took place, and information about the operations that were performed as part of the activity. Activity reports are typically delivered to a reply-to queue where they are collected together. By studying the activity reports related to a message, you can determine the route that the message took through the queue manager network.

Activity report usage

When messages are routed through a queue manager network, activity reports can be generated. You can use activity report information in the following ways:

Determine the last known location of a message

If a message that is enabled for activity recording does not reach its intended destination, activity reports generated for the message as it was routed through a queue manager network can be studied to determine the last known location of the message.

Determine configuration issues with a queue manager network

A number of messages enabled for activity recording can be sent into a queue manager network. By studying the activity reports related to each message it can become apparent that they have not taken the expected route. There are many reasons why this can occur, for example, a channel could have stopped, forcing the message to take an alternative route. In these situations, a system administrator can determine whether there are any problems in the queue manager network, and if there are, correct them.

Note: You can use activity recording in conjunction with trace-route messages by using the IBM MQ display route application.

Activity report format

Activity reports are PCF messages generated by applications that have performed an activity on behalf of a message. Activity reports are standard IBM MQ report messages containing a message descriptor and message data, as follows:

The message descriptor

- An MQMD structure

Message data

- An embedded PCF header (MQEPH)
- Activity report message data

Activity report message data consists of the *Activity* PCF group, and if generated for a trace-route message, the *TraceRoute* PCF group.

Related information:

MQMD - Message descriptor

MQEPH - Embedded PCF header

Controlling activity recording:

Enable activity recording at the queue manager level. To enable an entire queue manager network, individually enable every queue manager in the network for activity recording. If you enable more queue managers, more activity reports are generated.

About this task

To generate activity reports for a message as it is routed through a queue manager: define the message to request activity reports; enable the queue manager for activity recording; and ensure that applications performing activities on the message are capable of generating activity reports.

If you do *not* want activity reports to be generated for a message as it is routed through a queue manager, *disable* the queue manager for activity recording.

Procedure

1. Request activity reports for a message
   a. In the message descriptor of the message, specify MQRO_ACTIVITY in the *Report* field.
   b. In the message descriptor of the message, specify the name of a reply-to queue in the *ReplyToQ* field.

Warning: Avoid enabling all messages in a queue manager network for activity recording. Messages enabled for activity recording can have many activity reports generated on their behalf. If every message in a queue manager network is enabled for activity recording, the queue manager network traffic can increase to an unacceptable level.

2. Enable or disable the queue manager for activity recording. Use the MQSC command `ALTER QMGR`, specifying the parameter `ACTIVREC`, to change the value of the queue manager attribute. The value can be:

   MSG The queue manager is enabled for activity recording. Any activity reports generated are delivered to the reply-to queue specified in the message descriptor of the message. This is the default value.

   QUEUE
   The queue manager is enabled for activity recording. Any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE. The system queue can also be used to forward activity reports to a common queue.

   DISABLED
   The queue manager is disabled for activity recording. No activity reports are generated while in the scope of this queue manager.

   For example, to enable a queue manager for activity recording and specify that any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE, use the following MQSC command:

   ```
   ALTER QMGR ACTIVREC(QUEUE)
   ```

   Remember: When you modify the *ACTIVREC* queue manager attribute, a running MCA does not detect the change until the channel is restarted.

3. Ensure that your application uses the same algorithm as MCAs use to determine whether to generate an activity report for a message:
   a. Verify that the message has requested activity reports to be generated
   b. Verify that the queue manager where the message currently resides is enabled for activity recording
   c. Put the activity report on the queue determined by the *ACTIVREC* queue manager attribute

Setting up a common queue for activity reports:

To determine the locations of the activity reports related to a specific message when the reports are delivered to the local system queue, it is more efficient to use a common queue on a single node

Before you begin

Set the `ACTIVREC` parameter to enable the queue manager for activity recording and to specify that any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE.

About this task

If a number of queue managers in a queue manager network are set to deliver activity reports to the local system queue, it can be time consuming to determine the locations of the activity reports related to a specific message. Alternatively, use a single node, which is a queue manager that hosts a common queue. All the queue managers in a queue manager network can deliver activity reports to this common queue. The benefit of using a common queue is that queue managers do not have to deliver activity reports to the reply-to queue specified in a message and, when determining the locations of the activity reports related to a message, you query one queue only.

To set up a common queue, perform the following steps:

Procedure

1. Select or define a queue manager as the single node
2. On the single node, select or define a queue for use as the common queue
3. On all queue managers where activity reports are to be delivered to the common queue, redefine the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE as a remote queue definition:
   a. Specify the name of the single node as the remote queue manager name
   b. Specify the name of the common queue as the remote queue name

Determining message route information:

To determine a message route, obtain the information from the activity reports collected. Determine whether enough activity reports are on the reply-to queue to enable you to determine the required information and arrange the activity reports in order.

About this task

The order that activity reports are put on the reply-to queue does not necessarily correlate to the order in which the activities were performed. You must order activity reports manually, unless they are generated for a trace-route message, in which case you can use the IBM MQ display route application to order the activity reports.

Determine whether enough activity reports are on the reply-to queue for you to obtain the necessary information:

Procedure

1. Identify all related activity reports on the reply-to queue by comparing identifiers of the activity reports and the original message. Ensure you set the report option of the original message such that the activity reports can be correlated with the original message.
2. Order the identified activity reports from the reply-to queue. You can use the following parameters from the activity report:

   OperationType

   > The types of operations performed might enable you to determine the activity report that was generated directly before, or after, the current activity report.
   >
   > For example, an activity report details that an MCA sent a message from a transmission queue down a channel. The last operation detailed in the activity report has an *OperationType* of **send** and details that the message was sent using the channel, CH1, to the destination queue manager, QM1. This means that the next activity performed on the message will have occurred on queue manager, QM1, and that it will have begun with a **receive** operation from channel, CH1. By using this information you can identify the next activity report, providing it exists and has been acquired.

   OperationDate **and** *OperationTime*

   > You can determine the general order of the activities from the dates and times of the operations in each activity report.
   >
   > **Warning:** Unless every queue manager in the queue manager network has their system clocks synchronized, ordering by date and time does not guarantee that the activity reports are in the correct sequence. You must establish the order manually.

   The order of the activity reports represents the route, or partial route, that the message took through the queue manager network.
3. Obtain the information you need from the activity information in the ordered activity reports. If you have insufficient information about the message, you might be able to acquire further activity reports.

Retrieving further activity reports:

To determine a message route, sufficient information must be available from the activity reports collected. If you retrieve the activity reports related to a message from the reply-to queue that the message specified, but you not have the necessary information, look for further activity reports.

About this task

To determine the locations of any further activity reports, perform the following steps:

Procedure

1. For any queue managers in the queue manager network that deliver activity reports to a common queue, retrieve activity reports from the common queue that have a *CorrelId* that matches the *MsgId* of the original message.
2. For any queue managers in the queue manager network that do not deliver activity reports to a common queue, retrieve activity reports as follows:
   a. Examine the existing activity reports to identify queue managers through which the message was routed.
   b. For these queue managers, identify the queue managers that are enabled for activity recording.
   c. For these queue managers, identify any that did not return activity reports to the specified reply-to queue.
   d. For each of the queue managers that you identify, check the system queue SYSTEM.ADMIN.ACTIVITY.QUEUE and retrieve any activity reports that have a *CorrelId* that matches the *MsgId* of the original message.
   e. If you find no activity reports on the system queue, check the queue manager dead letter queue, if one exists. An activity report can only be delivered to a dead letter queue if the report option, MQRO_DEAD_LETTER_Q, is set.
3. Arrange all the acquired activity reports in order. The order of the activity reports then represents the route, or partial route, that the message took.
4. Obtain the information you need from the activity information in the ordered activity reports. In some circumstances, recorded activity information cannot reach the specified reply-to queue, a common queue, or a system queue.

Circumstances where activity information is not acquired:

To determine the complete sequence of activities performed on behalf of a message, information related to every activity must be acquired. If the information relating to any activity has not been recorded, or has not been acquired, you can determine only a partial sequence of activities.

Activity information is not recorded in the following circumstances:
- The message is processed by an IBM MQ queue manager earlier than Version 6.0.
- The message is processed by a queue manager that is not enabled for activity recording.
- The application that expected to process the message is not running.

Recorded activity information is unable to reach the specified reply-to queue in the following circumstances:
- There is no channel defined to route activity reports to the reply-to queue.
- The channel to route activity reports to the reply-to queue is not running.
- The remote queue definition to route activity reports back to the queue manager where the reply-to queue resides (the queue manager alias), is not defined.
- The user that generated the original message does not have open, or put, authority to the queue manager alias.

- The user that generated the original message does not have open, or put, authority to the reply-to queue.
- The reply-to queue is put inhibited.

Recorded activity information is unable to reach the system queue, or a common queue, in the following circumstances:
- If a common queue is to be used and there is no channel defined to route activity reports to the common queue.
- If a common queue is to be used and the channel to route activity reports to the common queue is not running.
- If a common queue is to be used and the system queue is incorrectly defined.
- The user that generated the original message does not have open, or put, authority to the system queue.
- The system queue is put inhibited.
- If a common queue is to be used and the user that generated the original message does not have open, or put, authority to the common queue.
- If a common queue is to be used and the common queue is put inhibited.

In these circumstances, providing the activity report does not have the report option MQRO_DISCARD_MSG specified, the activity report can be retrieved from a dead letter queue if one was defined on the queue manager where the activity report was rejected. An activity report will only have this report option specified if the original message, from which the activity report was generated, had both MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG specified in the Report field of the message descriptor.

Trace-route messaging

Trace-route messaging is a technique that uses *trace-route messages* to record activity information for a message. Trace-route messaging involves sending a trace-route message into a queue manager network.

As the trace-route message is routed through the queue manager network, activity information is recorded. This activity information includes information about the applications that performed the activities, when they were performed, and the operations that were performed as part of the activities. You can use the information recorded using trace-route messaging for the following purposes:

To determine the last known location of a message

If a message does not reach its intended destination, you can use the activity information recorded for a trace-route message to determine the last known location of the message. A trace-route message is sent into a queue manager network with the same target destination as the original message, intending that it follows the same route. Activity information can be accumulated in the message data of the trace-route message, or recorded using activity reports. To increase the probability that the trace-route message follows the same route as the original message, you can modify the trace-route message to mimic the original message.

To determine configuration issues with a queue manager network

Trace-route messages are sent into a queue manager network and activity information is recorded. By studying the activity information recorded for a trace-route message, it can become apparent that the trace-route message did not follow the expected route. There are many reasons why this can occur, for example, a channel might be inactive, forcing the message to take an alternative route. In these situations, a system administrator can determine whether there are any problems in the queue manager network, and if there are, correct them.

You can use the IBM MQ display route application to configure, generate, and put trace-route messages into a queue manager network.

Warning: If you put a trace-route message to a distribution list, the results are undefined.

Related concepts:

"Trace-route message reference" on page 1048
Use this page to obtain an overview of the trace-route message format. The trace-route message data includes parameters that describe the activities that the trace-route message has caused

How activity information is recorded:

With trace-route messaging, you can record activity information in the message data of the trace-route message, or use activity reports. Alternatively, you can use both techniques.

Accumulating activity information in the message data of the trace-route message

As a trace-route message is routed through a queue manager network, information about the activities performed on behalf of the trace-route message can be accumulated in the message data of the trace-route message. The activity information is stored in *Activity* PCF groups. For every activity performed on behalf of the trace-route message, an *Activity* PCF group is written to the end of the PCF block in the message data of the trace-route message.

Additional activity information is recorded in trace-route messaging, in a PCF group called the *TraceRoute* PCF group. The additional activity information is stored in this PCF group, and can be used to help determine the sequence of recorded activities. This technique is controlled by the *Accumulate* parameter in the *TraceRoute* PCF group.

Recording activity information using activity reports

As a trace-route message is routed through a queue manager network, an activity report can be generated for every activity that was performed on behalf of the trace-route message. The activity information is stored in the *Activity* PCF group. For every activity performed on behalf of a trace-route message, an activity report is generated containing an *Activity* PCF group. Activity recording for trace-route messages works in the same way as for any other message.

Activity reports generated for trace-route messages contain additional activity information compared to the those generated for any other message. The additional information is returned in a *TraceRoute* PCF group. The information contained in the *TraceRoute* PCF group is accurate only from the time the activity report was generated. You can use the additional information to help determine the sequence of activities performed on behalf of the trace-route message.

Acquiring recorded activity information:

When a trace-route message has reached its intended destination, or is discarded, the method that you use to acquire the activity information depends on how that information was recorded.

Before you begin

If you are unfamiliar with activity information, refer to "How activity information is recorded."

About this task

Use the following methods to acquire the activity information after the trace-route message has reached its intended destination, or is discarded:

Procedure

- Retrieve the trace-route message. The *Deliver* parameter, in the *TraceRoute* PCF group, controls whether a trace-route message is placed on the target queue on arrival, or whether it is discarded. If the trace-route message is delivered to the target queue, you can retrieve the trace-route message from this queue. Then, you can use the IBM MQ display route application to display the activity information.

To request that activity information is accumulated in the message data of a trace-route message, set the *Accumulate* parameter in the *TraceRoute* PCF group to `MQROUTE_ACCUMULATE_IN_MSG`.

- Use a trace-route reply message. When a trace-route message reaches its intended destination, or the trace-route message cannot be routed any further in a queue manager network, a trace-route reply message can be generated. A trace-route reply message contains a duplicate of all the activity information from the trace-route message, and is either delivered to a specified reply-to queue, or the system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE. You can use the IBM MQ display route application to display the activity information.

  To request a trace-route reply message, set the *Accumulate* parameter in the *TraceRoute* PCF group to `MQROUTE_ACCUMULATE_AND_REPLY`.
- Use activity reports. If activity reports are generated for a trace-route message, you must locate the activity reports before you can acquire the activity information. Then, to determine the sequence of activities, you must order the activity reports.

Controlling trace-route messaging:

Enable trace-route messaging at the queue manager level, so that applications in the scope of that queue manager can write activity information to a trace-route message. To enable an entire queue manager network, individually enable every queue manager in the network for trace-route messaging. If you enable more queue managers, more activity reports are generated.

Before you begin

If you are using activity reports to record activity information for a trace-route message, refer to "Controlling activity recording" on page 984.

About this task

To record activity information for a trace-route message as it is routed through a queue manager, perform the following steps:

Procedure

- Define how activity information is to be recorded for the trace-route message. Refer to "Generating and configuring a trace-route message" on page 993
- If you want to accumulate activity information in the trace-route message, ensure that the queue manager is enabled for trace-route messaging
- If you want to accumulate activity information in the trace-route message, ensure that applications performing activities on the trace-route message are capable of writing activity information to the message data of the trace-route message

Related concepts:

"Generating and configuring a trace-route message" on page 993
A trace-route message comprises specific message descriptor and message data parts. To generate a trace-route message, either create the message manually or use the IBM MQ display route application.

Related tasks:

"Controlling activity recording" on page 984
Enable activity recording at the queue manager level. To enable an entire queue manager network, individually enable every queue manager in the network for activity recording. If you enable more queue managers, more activity reports are generated.

Enabling queue managers for trace-route messaging:

To control whether queue managers are enabled or disabled for trace-route messaging use the queue manager attribute ROUTEREC.

Use the MQSC command `ALTER QMGR`, specifying the parameter `ROUTEREC` to change the value of the queue manager attribute. The value can be any of the following values:

MSG The queue manager is enabled for trace-route messaging. Applications within the scope of the queue manager can write activity information to the trace-route message.

If the *Accumulate* parameter in the *TraceRoute* PCF group is set as `MQROUTE_ACCUMULATE_AND_REPLY`, and the next activity to be performed on the trace-route message:

- is a discard
- is a put to a local queue (target queue or dead-letter queue)
- will cause the total number of activities performed on the trace-route message to exceed the value of parameter the *MaxActivities*, in the *TraceRoute* PCF group .

a trace-route reply message is generated, and delivered to the reply-to queue specified in the message descriptor of the trace-route message.

QUEUE

The queue manager is enabled for trace-route messaging. Applications within the scope of the queue manager can write activity information to the trace-route message.

If the *Accumulate* parameter in the *TraceRoute* PCF group is set as `MQROUTE_ACCUMULATE_AND_REPLY`, and the next activity to be performed on the trace-route message:

- is a discard
- is a put to a local queue (target queue or dead-letter queue)
- will cause the total number of activities performed on the trace-route message to exceed the value of parameter the *MaxActivities*, in the *TraceRoute* PCF group .

a trace-route reply message is generated, and delivered to the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

DISABLED

The queue manager is disabled for trace-route messaging. Activity information is not accumulated in the the trace-route message, however the *TraceRoute* PCF group can be updated while in the scope of this queue manager.

For example, to disable a queue manager for trace-route messaging, use the following MQSC command:

```
ALTER QMGR ROUTEREC(DISABLED)
```

Remember: When you modify the *ROUTEREC* queue manager attribute, a running MCA does not detect the change until the channel is restarted.

Enabling applications for trace-route messaging:

To enable trace-route messaging for a user application, base your algorithm on the algorithm used by message channel agents (MCAs)

Before you begin

If you are not familiar with the format of a trace-route message, see "Trace-route message reference" on page 1048.

About this task

Message channel agents (MCAs) are enabled for trace-route messaging. To enable a user application for trace-route messaging, use the following steps from the algorithm that MCAs use:

Procedure

1. Determine whether the message being processed is a trace-route message. If the message does not conform to the format of a trace-route message, the message is not processed as a trace-route message.
2. Determine whether activity information is to be recorded. If the detail level of the performed activity is not less than the level of detail specified by the *Detail* parameter, activity information is recorded under specific circumstances. This information is only recorded if the trace-route message requests accumulation, and the queue manager is enabled for trace-route messaging, or if the trace-route message requests an activity report and the queue manager is enabled for activity recording.
   - If activity information is to be recorded, increment the *RecordedActivities* parameter.
   - If activity information is not to be recorded, increment the *UnrecordedActivities* parameter.
3. Determine whether the total number of activities performed on the trace-route message exceeds the value of the *MaxActivities* parameter.

   The total number of activities is the sum of *RecordedActivities*, *UnrecordedActivities*, and *DiscontinuityCount*.

   If the total number of activities exceeds *MaxActivities*, reject the message with feedback MQFB_MAX_ACTIVITIES.
4. If value of *Accumulate* is set as MQROUTE_ACCUMULATE_IN_MSG or MQROUTE_ACCUMULATE_AND_REPLY, and the queue manager is enabled for trace-route messaging, write an Activity PCF group to the end of the PCF block in the message data of a trace-route message.
5. Deliver the trace-route message to a local queue.
   - If the parameter, *Deliver*, is specified as MQROUTE_DELIVER_NO, reject the trace-route message with feedback MQFB_NOT_DELIVERED.
   - If the parameter, *Deliver*, is specified as MQROUTE_DELIVER_YES, deliver the trace-route message to the local queue.
6. Generate a trace-route reply message if all the following conditions are true:
   - The trace-route message was delivered to a local queue or rejected
   - The value of the parameter, *Accumulate*, is MQROUTE_ACCUMULATE_AND_REPLY
   - The queue manager is enabled for trace-route messaging

   The trace-route reply message is put on the queue determined by the ROUTEREC queue manager attribute.
7. If the trace-route message requested an activity report and the queue manager is enabled for activity recording, generate an activity report. The activity report is put on the queue determined by the ACTIVREC queue manager attribute.

Generating and configuring a trace-route message:

A trace-route message comprises specific message descriptor and message data parts. To generate a trace-route message, either create the message manually or use the IBM MQ display route application.

A trace-route message consists of the following parts:

Message descriptor
An MQMD structure, with the *Format* field set to MQFMT_ADMIN or MQFMT_EMBEDDED_PCF.

Message data
One of the following combinations:

- A PCF header (MQCFH) and trace-route message data, if *Format* is set to MQFMT_ADMIN
- An embedded PCF header (MQEPH), trace-route message data, and additional user-specified message data, if *Format* is set to MQFMT_EMBEDDED_PCF

The trace-route message data consists of the *TraceRoute* PCF group and one or more *Activity* PCF groups.

Manual generation

When generating a trace-route message manually, an *Activity* PCF group is not required. *Activity* PCF groups are written to the message data of the trace-route message when an MCA or user-written application performs an activity on its behalf.

The IBM MQ display route application

Use the IBM MQ display route application, **dspmqrte** , to configure, generate and put a trace-route message into a queue manager network. Set the *Format* parameter in the message descriptor to MQFMT_ADMIN. You cannot add user data to the trace-route message generated by the IBM MQ display route application.

Restriction: **dspmqrte** cannot be issued on queue managers before IBM WebSphere MQ Version 6.0 or on IBM MQ for z/OS queue managers. If you want the first queue manager the trace-route message is routed through to be a queue manager of this type, connect to the queue manager as a IBM WebSphere MQ Version 6.0 or later client using the optional parameter `-c`.

Mimicking the original message:

When using a trace-route message to determine the route another message has taken through a queue manager network, the more closely a trace-route message mimics the original message, the greater the chance that the trace-route message will follow the same route as the original message.

The following message characteristics can affect where a message is forwarded to within a queue manager network:

Priority
The priority can be specified in the message descriptor of the message.

Persistence
The persistence can be specified in the message descriptor of the message.

Expiration
The expiration can be specified in the message descriptor of the message.

Report options
Report options can be specified in the message descriptor of the message.

Message size

To mimic the size of a message, additional data can be written to the message data of the message. For this purpose, additional message data can be meaningless.

Tip: The IBM MQ display route application cannot specify message size.

Message data

Some queue manager networks use content based routing to determine where messages are forwarded. In these cases the message data of the trace-route message needs to be written to mimic the message data of the original message.

Tip: The IBM MQ display route application cannot specify message data.

The TraceRoute PCF group:

Attributes in the *TraceRoute* PCF group control the behavior of a trace-route message. The *TraceRoute* PCF group is in the message data of every trace-route message.

The following table lists the parameters in the *TraceRoute* group that an MCA recognizes. Further parameters can be added if user-written applications are written to recognize them, as described in "Additional activity information" on page 999.

Table 112. TraceRoute PCF group

Parameter	Type
TraceRoute	MQCFGR
Detail	MQCFIN
RecordedActivities	MQCFIN
UnrecordedActivities	MQCFIN
DiscontinuityCount	MQCFIN
MaxActivities	MQCFIN
Accumulate	MQCFIN
Forward	MQCFIN
Deliver	MQCFIN

Descriptions of each parameter in the *TraceRoute* PCF group follows:

Detail Specifies the detail level of activity information that is to be recorded. The value can be any of the following values:

MQROUTE_DETAIL_LOW

Only activities performed by user application are recorded.

MQROUTE_DETAIL_MEDIUM

Activities specified in MQROUTE_DETAIL_LOW should be recorded. Additionally, activities performed by MCAs are recorded.

MQROUTE_DETAIL_HIGH

Activities specified in MQROUTE_DETAIL_LOW, and MQROUTE_DETAIL_MEDIUM should be recorded. MCAs do not record any further activity information at this level of detail. This option is only available to user applications that are to record further activity information. For example, if a user application determines the route a message takes by considering certain message characteristics, the information about the routing logic could be included with this level of detail.

RecordedActivities

Specifies the number of recorded activities performed on behalf of the trace-route message. An activity is considered to be recorded if information about it has been written to the trace-route message, or if an activity report has been generated. For every recorded activity, *RecordedActivities* increments by one.

UnrecordedActivities
Specifies the number of unrecorded activities performed on behalf of the trace-route message. An activity is considered to be unrecorded if an application that is enabled for trace-route messaging neither accumulates, nor writes the related activity information to an activity report.

An activity performed on behalf of a trace-route message is unrecorded in the following circumstances:

- The detail level of the performed activity is less than the level of detail specified by the parameter *Detail*.
- The trace-route message requests an activity report but not accumulation, and the queue manager is not enabled for activity recording.
- The trace-route message requests accumulation but not an activity report, and the queue manager is not enabled for trace-route messaging.
- The trace-route message requests both accumulation and an activity report, and the queue manager is not enabled for activity recording and trace route messaging.
- The trace-route message requests neither accumulation nor an activity report.

For every unrecorded activity the parameter, *UnrecordedActivities*, increments by one.

DiscontinuityCount
Specifies the number of times the trace-route message has been routed through a queue manager with applications that were not enabled for trace-route messaging. This value is incremented by the queue manager. If this value is greater than 0, only a partial message route can be determined.

MaxActivities
Specifies the maximum number of activities that can be performed on behalf of the trace-route message.

The total number of activities is the sum of *RecordedActivities*, *UnrecordedActivities*, and *DiscontinuityCount*. The total number of activities must not exceed the value of *MaxActivities*.

The value of *MaxActivities* can be:

A positive integer
The maximum number of activities.

If the maximum number of activities is exceeded, the trace-route message is rejected with feedback MQFB_MAX_ACTIVITIES. This can prevent the trace-route message from being forwarded indefinitely if caught in an infinite loop.

MQROUTE_UNLIMITED_ACTIVITIES
An unlimited number of activities can be performed on behalf of the trace-route message.

Accumulate
Specifies the method used to accumulate activity information. The value can be any of the following values:

MQROUTE_ACCUMULATE_IN_MSG
If the queue manager is enabled for trace-route messaging, activity information is accumulated in the message data of the trace-route message.

If this value is specified, the trace-route message data consists of the following:

- The *TraceRoute* PCF group.
- Zero or more *Activity* PCF groups.

MQROUTE_ACCUMULATE_AND_REPLY
If the queue manager is enabled for trace-route messaging, activity information is accumulated in the message data of the trace-route message, and a trace-route reply message is generated if any of the following occur:

- The trace-route message is discarded by an IBM MQ Version 6 (or later) queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by an IBM MQ Version 6 (or later) queue manager.
- The number of activities performed on the trace-route message exceeds the value of *MaxActivities*.

If this value is specified, the trace-route message data consists of the following:
- The *TraceRoute* PCF group.
- Zero or more *Activity* PCF groups.

MQROUTE_ACCUMULATE_NONE
Activity information is not accumulated in the message data of the trace-route message.

If this value is specified, the trace-route message data consists of the following:
- The *TraceRoute* PCF group.

Forward
Specifies where a trace-route message can be forwarded to. The value can be:

MQROUTE_FORWARD_IF_SUPPORTED
The trace-route message is only forwarded to queue managers that will honor the value of the *Deliver* parameter from the *TraceRoute* group.

MQROUTE_FORWARD_ALL
The trace-route message is forwarded to any queue manager, regardless of whether the value of the *Deliver* parameter will be honored.

Queue managers use the following algorithm when determining whether to forward a trace-route message to a remote queue manager:
1. Determine whether the remote queue manager is capable of supporting trace-route messaging.
   - If the remote queue manager is capable of supporting trace-route messaging, the algorithm continues to step 4.
   - If the remote queue manager is not capable of supporting trace-route messaging, the algorithm continues to step 2
2. Determine whether the *Deliver* parameter from the *TraceRoute* group contains any unrecognized delivery options in the MQROUTE_DELIVER_REJ_UNSUP_MASK bit mask.
   - If any unrecognized delivery options are found, the trace-route message is rejected with feedback MQFB_UNSUPPORTED_DELIVERY.
   - If no unrecognized delivery options are found, the algorithm continues to step 3.
3. Determine the value of the parameter *Deliver* from the *TraceRoute* PCF group in the trace-route message.
   - If *Deliver* is specified as MQROUTE_DELIVER_YES, the trace-route message is forwarded to the remote queue manager.
   - If *Deliver* is specified as MQROUTE_DELIVER_NO, the algorithm continues to step 4.
4. Determine whether the *Forward* parameter from the *TraceRoute* group contains any unrecognized forwarding options in the MQROUTE_FORWARDING_REJ_UNSUP_MASK bit mask.
   - If any unrecognized forwarding options are found, the trace-route message is rejected with feedback MQFB_UNSUPPORTED_FORWARDING.
   - If no unrecognized forwarding options are found, the algorithm continues to step 5.
5. Determine the value of the parameter *Forward* from the *TraceRoute* PCF group in the trace-route message.

- If *Forward* is specified as MQROUTE_FORWARD_IF_SUPPORTED, the trace-route message is rejected with feedback MQFB_NOT_FORWARDED.
- If *Forward* is specified as MQROUTE_FORWARD_ALL, trace-route message can be forwarded to the remote queue manager.

Deliver
Specifies the action to be taken if the trace-route message reaches its intended destination. User-written applications must check this attribute before placing a trace-route message on its target queue. The value can be any of the following values:

MQROUTE_DELIVER_YES
On arrival, the trace-route message is put on the target queue. Any application performing a get operation on the target queue can retrieve the trace-route message.

MQROUTE_DELIVER_NO
On arrival, the trace-route message is not delivered to the target queue. The message is processed according to its report options.

Setting up a common queue for trace-route reply messages:

To determine the locations of the trace-route reply messages related to a specific message when the reports are delivered to the local system queue, it is more efficient to use a common queue on a single node

Before you begin

Set the `ROUTEREC` parameter to enable the queue manager for trace-route messaging and to specify that any trace-route reply messages generated are delivered to the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

About this task

If a number of queue managers in a queue manager network are set to deliver trace-route reply messages to the local system queue, it can be time consuming to determine the locations of the trace-route reply messages related to a specific message. Alternatively, use a single node, which is a queue manager that hosts a common queue. All the queue managers in a queue manager network can deliver trace-route reply messages to this common queue. The benefit of using a common queue is that queue managers do not have to deliver trace-route reply messages to the reply-to queue specified in a message and, when determining the locations of the trace-route reply messages related to a message, you query one queue only.

To set up a common queue, perform the following steps:

Procedure

1. Select or define a queue manager as the single node
2. On the single node, select or define a queue for use as the common queue
3. On all queue managers that forward trace-route reply messages to the common queue, redefine the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE as a remote queue definition
   a. Specify the name of the single node as the remote queue manager name
   b. Specify the name of the common queue as the remote queue name

Acquiring and using recorded information:

Use any of the following techniques to acquire recorded activity information for a trace-route message

Note that the circumstances in which activity information is not acquired apply also to trace-route reply messages.

Activity information is not recorded when a trace-route message is processed by a queue manager that is disabled for both activity recording and trace-route messaging.

Acquiring information from trace-route reply messages:

To acquire activity information you locate the trace-route reply message. Then you retrieve the message and analyze the activity information.

About this task

You can acquire activity information from a trace-route reply message only if you know the location of the trace-route reply message. Locate the message and process the activity information as follows:

Procedure

1. Check the reply-to queue that was specified in the message descriptor of the trace-route message. If the trace-route reply message is not on the reply-to queue, check the following locations:
   - The local system queue, SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, on the target queue manager of the trace-route message
   - The common queue, if you have set up a common queue for trace-route reply messages
   - The local system queue, SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, on any other queue manager in the queue manager network, which can occur if the trace-route message has been put to a dead-letter queue, or the maximum number of activities was exceeded
2. Retrieve the trace-route reply message
3. Use the IBM MQ display route application to display the recorded activity information
4. Study the activity information and obtain the information that you need

Acquiring information from trace-route messages:

To acquire activity information you locate the trace-route message, which must have the appropriate parameters in the *TraceRoute* PCF group. Then you retrieve the message and analyze the activity information.

About this task

You can acquire activity information from a trace-route message only if you know the location of the trace-route message and it has the parameter *Accumulate* in the *TraceRoute* PCF group specified as either `MQROUTE_ACCUMULATE_IN_MSG` or `MQROUTE_ACCUMULATE_AND_REPLY`.

For the trace-route message to be delivered to the target queue the *Deliver* parameter in the *TraceRoute* PCF group must be specified as `MQROUTE_DELIVER_YES`.

Procedure

1. Check the target queue. If the trace-route message is not on the target queue, you can try to locate the trace-route message using a trace-route message enabled for activity recording. With the generated activity reports try to determine the last known location of the trace-route message.
2. Retrieve the trace-route message
3. Use the IBM MQ display route application to display the recorded activity information

4. Study the activity information and obtain the information that you need

Acquiring information from activity reports:

To acquire activity information you locate the activity report, which must have the report option specified in the message descriptor. Then you retrieve the activity report and analyze the activity information.

About this task

You can acquire activity information from an activity report only if you know the location of the activity report and the report option MQRO_ACTIVITY was specified in the message descriptor of the trace-route message.

Procedure

1. Locate and order the activity reports generated for a trace-route message. When you have located the activity reports, you can order them manually or use the IBM MQ display route application to order and display the activity information automatically.
2. Study the activity information and obtain the information that you need

Additional activity information:

As a trace-route message is routed through a queue manager network, user applications can record additional information by including one or more additional PCF parameters when writing the *Activity* group to the message data of the trace-route message or activity report.

Additional activity information can help system administrators to identify the route taken by a trace-route message took, or why that route was taken.

If you use the IBM MQ display route application to display the recorded information for a trace-route message, any additional PCF parameters can only be displayed with a numeric identifier, unless the parameter identifier of each parameter is recognized by the IBM MQ display route application. To recognize a parameter identifier, additional information must be recorded using the following PCF parameters. Include these PCF parameters in an appropriate place in the *Activity* PCF group.

GroupName

Table 113. Group name

Description	**Grouped parameters specifying the additional information.**
Identifier	MQGACF_VALUE_NAMING.
Data type	MQCFGR
Parameters in group	*ParameterName* *ParameterValue*

ParameterName

Table 114. Parameter name

Description	**Contains the name to be displayed by the IBM MQ display route application, which puts the value of *ParameterValue* into context.**
Identifier	MQCA_VALUE_NAME.
Data type	MQCFST
Included in PCF group:	*GroupName*.
Value:	The name to be displayed.

ParameterValue

Table 115. Parameter value

Description	**Contains the value to be displayed by the IBM MQ display route application.**
Identifier:	The PCF structure identifier for the additional information.
Data type:	The PCF structure data type for the additional information.
Included in PCF group:	*GroupName*.
Value:	The value to be displayed.

Examples of recording additional activity information

The following examples illustrate how a user application can record additional information when performing an activity on behalf of a trace-route message. In both examples, the IBM MQ display route application is used to generate a trace-route message, and display the activity information returned to it.

Example 1:

Additional activity information is recorded by a user application in a format where the parameter identifier *is not* recognized by the IBM MQ display route application.

1. The IBM MQ display route application is used to generate and put a trace-route message into a queue manager network. The necessary options are set to request the following:
   - Activity information is accumulated in the message data of the trace-route message.
   - On arrival at the target queue the trace-route message is discarded, and a trace-route reply message is generated and delivered to a specified reply-to queue.
   - On receipt of the trace-route reply message, the IBM MQ display route application displays the accumulated activity information.

   The trace-route message is put into the queue manager network.
2. As the trace-route message is routed through the queue manager network a user application, that is enabled for trace-route messaging, performs a low detail activity on behalf of the message. In addition to writing the standard activity information to the trace-route message, the user application writes the following PCF parameter to the end of the Activity group:

   ColorValue

   Identifier
   65536

   Data type
   MQCFST

   Value 'Red'

This additional PCF parameter gives further information about the activity that was performed, however it is written in a format where the parameter identifier *is not* recognized by the IBM MQ display route application.

3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the IBM MQ display route application. The additional activity information is displayed as follows:

   ```
   65536: 'Red'
   ```

   The IBM MQ display route application does not recognize the parameter identifier of the PCF parameter and displays it as a numeric value. The context of the additional information is not clear.

   For an example of when the IBM MQ display route application does recognize the parameter identifier of the PCF parameter, see "Example 2."

Example 2:

Additional activity information is recorded by a user application in a format where the parameter identifier *is* recognized by the IBM MQ display route application.

1. The IBM MQ display route application is used to generate and put a trace-route message into a queue manager network in the same fashion as in "Example 1" on page 1000.
2. As the trace-route message is routed through the queue manager network a user application, that is enabled for trace-route messaging, performs a low detail activity on behalf of the message. In addition to writing the standard activity information to the trace-route message, the user application writes the following PCF parameters to the end of the Activity group:

   ColorInfo

Table 116. Color information

Description	**Grouped parameters specifying information about a color.**
Identifier:	MQGACF_VALUE_NAMING.
Data type:	MQCFGR.
Parameters in group:	*ColorName* *ColorValue*

   ColorName

Table 117. Color name

Description	**Contains the name to be displayed by the IBM MQ display route application which puts the value of** *ColorValue* **into context.**
Identifier:	MQCA_VALUE_NAME.
Data type:	MQCFST.
Included in PCF group:	*ColorInfo*.
Value:	'Color'

   ColorValue

Table 118. Color value

Description	Contains the value to be displayed by the IBM MQ display route application.
Identifier:	65536.
Data type:	MQCFST.
Included in PCF group:	*ColorInfo*.
Value:	'Red'

These additional PCF parameters gives further information about the activity that was performed. These PCF parameters are written in a format where the parameter identifier *is* recognized by the IBM MQ display route application.

3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the IBM MQ display route application. The additional activity information is displayed as follows:

```
Color: 'Red'
```

The IBM MQ display route application recognizes that the parameter identifier of the PCF structure containing the value of the additional activity information has a corresponding name. The corresponding name is displayed instead of the numeric value.

IBM MQ display route application

Use the IBM MQ display route application (**dspmqrte**) to work with trace-route messages and activity information related to a trace-route message, using a command-line interface. z/OS The IBM MQ display route application is not available for IBM MQ for z/OS queue managers.

You can use the IBM MQ display route application for the following purposes:

- To configure, generate, and put a trace-route message into a queue manager network.

  By putting a trace-route message into a queue manager network, activity information can be collected and used to determine the route that the trace-route message took. You can specify the characteristics of the trace-route messages as follows:
  - The destination of the trace-route message.
  - How the trace-route message mimics another message.
  - How the trace-route message should be handled as it is routed through a queue manager network.
  - Whether activity recording or trace-route messaging are used to record activity information.
- To order and display activity information related to a trace-route message.

  If the IBM MQ display route application has put a trace-route message into a queue manager network, after the related activity information has been returned, the information can be ordered and displayed immediately. Alternatively, the IBM MQ display route application can be used to order, and display, activity information related to a trace-route message that was previously generated.

Related information:
dspmqrte

Parameters for trace-route messages:

Use this page to obtain an overview of the parameters provided by the IBM MQ display route application, **dspmqrte**, to determine the characteristics of a trace-route message, including how it is treated as it is routed through a queue manager network.

Related information:
dspmqrte

Queue manager connection:

Use this page to specify the queue manager that the IBM MQ display route application connects to

-c Specifies that the IBM MQ display route application connects as a client application.

If you do not specify this parameter, the IBM MQ display route application does not connect as a client application.

-m *QMgrName*
The name of the queue manager to which the IBM MQ display route application connects. The name can contain up to 48 characters.

If you do not specify this parameter, the default queue manager is used.

The target destination:

Use this page to specify the target destination of a trace-route message

-q *TargetQName*
If the IBM MQ display route application is being used to send a trace-route message into a queue manager network, *TargetQName* specifies the name of the target queue.

-ts *TargetTopicString*
Specifies the topic string.

-qm *TargetQMgr*
Qualifies the target destination; normal queue manager name resolution will then apply. The target destination is specified with *-q TargetQName* or *-ts TargetTopicString* .

If you do not specify this parameter, the queue manager to which the IBM MQ display route application is connected is used as the target queue manager.

-o Specifies that the target destination is not bound to a specific destination. Typically this parameter is used when the trace-route message is to be put across a cluster. The target destination is opened with option MQOO_BIND_NOT_FIXED.

If you do not specify this parameter, the target destination is bound to a specific destination.

The publication topic:

For publish/subscribe applications, use this page to specify the topic string of a trace-route message for the IBM MQ display route application to publish

-ts *TopicName*
: Specifies a topic string to which the IBM MQ display route application is to publish a trace-route message, and puts this application into topic mode. In this mode, the application traces all of the messages that result from the publish request.

You can also use the IBM MQ display route application to display the results from an activity report that was generated for publish messages.

Message mimicking:

Use this page to configure a trace-route message to mimic a message, for example when the original message did not reach its intended destination

One use of trace-route messaging is to help determine the last known location of a message that did not reach its intended destination. The IBM MQ display route application provides parameters that can help configure a trace-route message to mimic the original message. When mimicking a message, you can use the following parameters:

-l *Persistence*

Specifies the persistence of the generated trace-route message. Possible values for *Persistence* are:

yes The generated trace-route message is persistent. (MQPER_PERSISTENT).

no The generated trace-route message is **not** persistent. (MQPER_NOT_PERSISTENT).

q The generated trace-route message inherits its persistence value from the destination specified by *-q TargetQName* or *-ts TargetTopicString*. (MQPER_PERSISTENCE_AS_Q_DEF).

A trace-route reply message, or any report messages, returned will share the same persistence value as the original trace-route message.

If *Persistence* is specified as **yes**, you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

If you do not specify this parameter, the generated trace-route message is **not** persistent.

-p *Priority*

Specifies the priority of the trace-route message. The value of *Priority* is either greater than or equal to 0, or MQPRI_PRIORITY_AS_Q_DEF. MQPRI_PRIORITY_AS_Q_DEF specifies that the priority value is taken from the destination specified by *-q TargetQName* or *-ts TargetTopicString*.

If you do not specify this parameter, the priority value is taken from the destination specified by *-q TargetQName* or *-ts TargetTopicString*.

-xs *Expiry*

Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-ro none | *ReportOption*

none Specifies no report options are set.

ReportOption

Specifies report options for the trace-route message. Multiple report options can be specified using a comma as a separator. Possible values for *ReportOption* are:

activity

The report option MQRO_ACTIVITY is set.

coa The report option MQRO_COA_WITH_FULL_DATA is set.

cod The report option MQRO_COD_WITH_FULL_DATA is set.

exception
The report option MQRO_EXCEPTION_WITH_FULL_DATA is set.

expiration
The report option MQRO_EXPIRATION_WITH_FULL_DATA is set.

discard
The report option MQRO_DISCARD_MSG is set.

If neither `-ro ReportOption` nor `-ro none` are specified, then the MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified.

The IBM MQ display route application does not allow you to add user data to the trace-route message. If you require user data to be added to the trace-route message you must generate the trace-route message manually.

Recorded activity information:

Use this page to specify the method used to return recorded activity information, which you can then use to determine the route that a trace-route message has taken

Recorded activity information can be returned as follows:
- In activity reports
- In a trace-route reply message
- In the trace-route message itself (having been put on the target queue)

When using **`dspmqrte`**, the method used to return recorded activity information is determined using the following parameters:

`-ro` *`activity`*
Specifies that activity information is returned using activity reports. By default activity recording is enabled.

`-ac -ar`
Specifies that activity information is accumulated in the trace-route message, and that a trace-route reply message is to be generated.

`-ac`
Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is **not** accumulated within the trace-route message.

`-ar`
Requests that a trace-route reply message containing all accumulated activity information is generated in the following circumstances:
- The trace-route message is discarded by an IBM MQ queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by an IBM MQ queue manager.
- The number of activities performed on the trace-route message exceeds the value of specified in *-s Activities*.

`-ac -d yes`
Specifies that activity information is accumulated in the trace-route message, and that on arrival, the trace-route message will be put on the target queue.

-ac
: Specifies that activity information is to be accumulated within the trace-route message.

  If you do not specify this parameter, activity information is **not** accumulated within the trace-route message.

-d yes
: On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.

  If you do not specify this parameter, the trace-route message is **not** put to the target queue.

The trace-route message can then be retrieved from the target queue, and the recorded activity information acquired.

You can combine these methods as required.

Additionally, the detail level of the recorded activity information can be specified using the following parameter:

-t *Detail*
: Specifies the activities that are recorded. The possible values for *Detail* are:

low Activities performed by user-defined application are recorded only.

medium

Activities specified in **low** are recorded. Additionally, publish activities and activities performed by MCAs are recorded.

high

Activities specified in **low**, and **medium** are recorded. MCAs do not expose any further activity information at this level of detail. This option is available to user-defined applications that are to expose further activity information only. For example, if a user-defined application determines the route a message takes by considering certain message characteristics, the routing logic could be included with this level of detail.

If you do not specify this parameter, medium level activities are recorded.

By default the IBM MQ display route application uses a temporary dynamic queue to store the returned messages. When the IBM MQ display route application ends, the temporary dynamic queue is closed, and any messages are purged. If the returned messages are required beyond the current execution of the IBM MQ display route application ends, then a permanent queue must be specified using the following parameters:

-rq *ReplyToQ*
: Specifies the name of the reply-to queue that all responses to the trace-route message are sent to. If the trace-route message is persistent, or if the *-n* parameter is specified, a reply-to queue must be specified that is **not** a temporary dynamic queue.

  If you do not specify this parameter then a dynamic reply-to queue is created using the system default model queue, SYSTEM.DEFAULT.MODEL.QUEUE.

-rqm *ReplyToQMgr*
: Specifies the name of the queue manager where the reply-to queue resides. The name can contain up to 48 characters.

  If you do not specify this parameter, the queue manager to which the IBM MQ display route application is connected is used as the reply-to queue manager.

How the trace-route message is handled:

Use this page to control how a trace-route message is handled as it is routed through a queue manager network.

The following parameters can restrict where the trace-route message can be routed in the queue manager network:

-d *Deliver*
: Specifies whether the trace-route message is to be delivered to the target queue on arrival. Possible values for *Deliver* are:

yes	On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.
no	On arrival, the trace-route message is **not** put to the target queue.

If you do not specify this parameter, the trace-route message is **not** put to the target queue.

-f *Forward*
: Specifies the type of queue manager that the trace-route message can be forwarded to. For details of the algorithm that queue managers use to determine whether to forward a message to a remote queue manager, refer to "The TraceRoute PCF group" on page 994. The possible values for *Forward* are:

all The trace-route message is forwarded to any queue manager.

Warning: If forwarded to an IBM MQ queue manager earlier than Version 6.0, the trace-route message will not be recognized and can be delivered to a local queue despite the value of the *-d Deliver* parameter.

supported
: The trace-route message is only forwarded to a queue manager that will honor the *Deliver* parameter from the *TraceRoute* PCF group

If you do not specify this parameter, the trace-route message will only be forwarded to a queue manager that will honor the *Deliver* parameter.

The following parameters can prevent a trace-route message from remaining in a queue manager network indefinitely:

-s *Activities*
: Specifies the maximum number of recorded activities that can be performed on behalf of the trace-route message before it is discarded. This prevents the trace-route message from being forwarded indefinitely if caught in an infinite loop. The value of *Activities* is either greater than or equal to 1, or MQROUTE_UNLIMITED_ACTIVITIES. MQROUTE_UNLIMITED_ACTIVITIES specifies that an unlimited number of activities can be performed on behalf of the trace-route message.

If you do not specify this parameter, an unlimited number of activities can be performed on behalf of the trace-route message.

-xs *Expiry*
: Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-xp *PassExpiry*
: Specifies whether the expiry time from the trace-route message is passed on to a trace-route reply message. Possible values for *PassExpiry* are:

yes The report option MQRO_PASS_DISCARD_AND_EXPIRY is specified in the message descriptor of the trace-route message.

If a trace-route reply message, or activity reports, are generated for the trace-route message, the MQRO_DISCARD report option (if specified), and the remaining expiry time are passed on.

This is the default value.

no The report option MQRO_PASS_DISCARD_AND_EXPIRY is not specified.

If a trace-route reply message is generated for the trace-route message, the discard option and expiry time from the trace-route message are **not** passed on.

If you do not specify this parameter, MQRO_PASS_DISCARD_AND_EXPIRY is not specified.

`-ro` *`discard`*
Specifies the MQRO_DISCARD_MSG report option. This can prevent the trace-route message remaining in the queue manager network indefinitely.

Display of activity information:

The IBM MQ display route application can display activity information for a trace-route message that it has just put into a queue manager network, or it can display activity information for a previously generated trace-route message. It can also display additional information recorded by user-written applications.

To specify whether activity information returned for a trace-route message is displayed, specify the following parameter:

`-n` Specifies that activity information returned for the trace-route message is not to be displayed.

If this parameter is accompanied by a request for a trace-route reply message, (*`-ar`*), or any of the report generating options from (*`-ro ReportOption`*), then a specific (non-model) reply-to queue must be specified using *`-rq ReplyToQ`* . By default, only activity report messages are requested.

After the trace-route message is put to the specified target queue, a 48 character hexadecimal string is displayed containing the message identifier of the trace-route message. The message identifier can be used by the IBM MQ display route application to display the activity information for the trace-route message at a later time, using the *-i CorrelId* parameter.

If you do not specify this parameter, activity information returned for the trace-route message is displayed in the form specified by the *-v* parameter.

When displaying activity information for a trace-route message that has just been put into a queue manager network, the following parameter can be specified:

`-w` *`WaitTime`*
Specifies the time, in seconds, that the IBM MQ display route application will wait for activity reports, or a trace-route reply message, to return to the specified reply-to queue.

If you do not specify this parameter, the wait time is specified as the expiry time of the trace-route message, plus 60 seconds.

When displaying previously accumulated activity information the following parameters must be set:

`-q` *`TargetQName`*
If the IBM MQ display route application is being used to view previously gathered activity information, *TargetQName* specifies the name of the queue where the activity information is stored.

`-i` *`CorrelId`*
This parameter is used when the IBM MQ display route application is used to display previously accumulated activity information only. There can be many activity reports and trace-route reply

messages on the queue specified by *-q TargetQName*. *CorrelId* is used to identify the activity reports, or a trace-route reply message, related to a trace-route message. Specify the message identifier of the original trace-route message in *CorrelId*.

The format of *CorrelId* is a 48 character hexadecimal string.

The following parameters can be used when displaying previously accumulated activity information, or when displaying current activity information for a trace-route message:

-b Specifies that the IBM MQ display route application will only browse activity reports or a trace-route reply message related to a message. This allows activity information to be displayed again at a later time.

If you do not specify this parameter, the IBM MQ display route application will destructively get activity reports or a trace-route reply message related to a message.

`-v summary | all | none | outline` *`DisplayOption`*

summary
The queues that the trace-route message was routed through are displayed.

all All available information is displayed.

none No information is displayed.

outline *DisplayOption*
Specifies display options for the trace-route message. Multiple display options can be specified using a comma as a separator.

If no values are supplied the following is displayed:

- The application name
- The type of each operation
- Any operation specific parameters

Possible values for *DisplayOption* are:

activity
All non-PCF group parameters in *Activity* PCF groups are displayed.

identifiers
Values with parameter identifiers MQBACF_MSG_ID or MQBACF_CORREL_ID are displayed. This overrides *msgdelta*.

message
All non-PCF group parameters in *Message* PCF groups are displayed. When this value is specified, you cannot specify *msgdelta*.

msgdelta
All non-PCF group parameters in *Message* PCF groups, that have changed since the last operation, are displayed. When this value is specified, you cannot specify *message*.

operation
All non-PCF group parameters in *Operation* PCF groups are displayed.

traceroute
All non-PCF group parameters in *TraceRoute* PCF groups are displayed.

If you do not specify this parameter, a summary of the message route is displayed.

Display of additional information

As a trace-route message is routed through a queue manager network, user-written applications can record additional information by writing one or more additional PCF parameters to the message data of the trace-route message or to the message data of an activity report. For the IBM MQ display route application to display additional information in a readable form it must be recorded in a specific format, as described in “Additional activity information” on page 999.

IBM MQ display route application examples:

The following examples show how you can use the IBM MQ display route application. In each example, two queue managers (QM1 and QM2) are inter-connected by two channels (QM2.TO.QM1 and QM1.TO.QM2).

Example 1 - Requesting activity reports:

Display activity information from a trace-route message delivered to the target queue

In this example the IBM MQ display route application connects to queue manager, QM1, and is used to generate and deliver a trace-route message to the target queue, TARGET.Q, on remote queue manager, QM2. The necessary report option is specified so that activity reports are requested as the trace-route reply message is routed. On arrival at the target queue the trace-route message is discarded. Activity information returned to the IBM MQ display route application using activity reports is put in order and displayed.

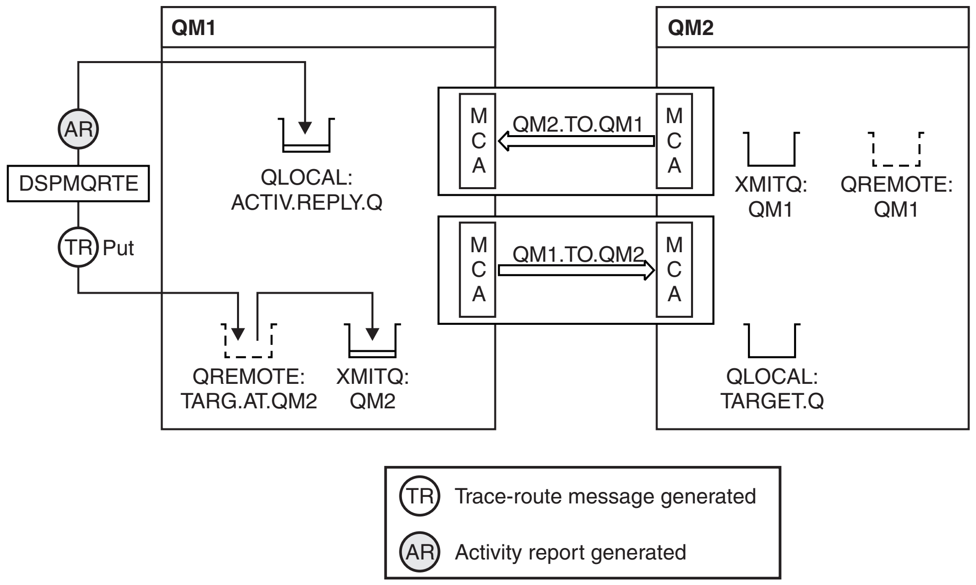


Figure 94. Requesting activity reports, Diagram 1

- The ACTIVREC attribute of each queue manager (QM1 and QM2) is set to MSG.
- The following command is issued:

  ```
  dspmqrte -m QM1 -q TARG.AT.QM2 -rq ACTIV.REPLY.Q
  ```

  QM1 is the name of the queue manager to which the IBM MQ display route application connects, TARG.AT.QM2 is the name of the target queue, and ACTIV.REPLY.Q is the name of the queue to which it is requested that all responses to the trace-route message are sent.

  Default values are assumed for all options that are not specified, but note in particular the -f option (the trace-route message is forwarded only to a queue manager that honors the Deliver parameter of the TraceRoute PCF group), the -d option (on arrival, the trace-route message is not put on the target queue), the -ro option (MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified), and the -t option (medium detail level activity is recorded).
- DSPMQRTE generates the trace-route message and puts it on the remote queue TARG.AT.QM2.

- DSPMQRTE then looks at the value of the ACTIVREC attribute of queue manager QM1. The value is MSG, therefore DSPMQRTE generates an activity report and puts it on the reply queue ACTIV.REPLY.Q.

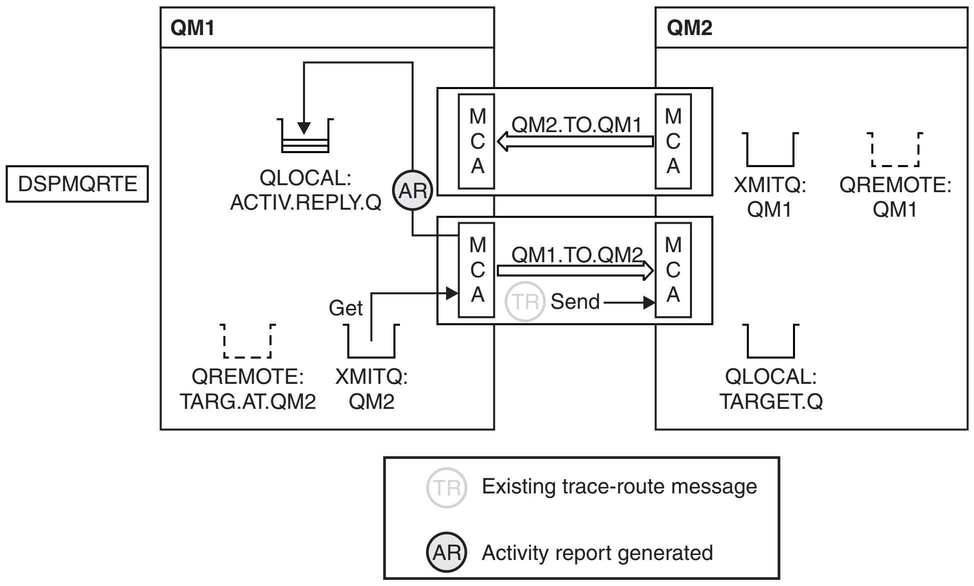


Figure 95. Requesting activity reports, Diagram 2

- The sending message channel agent (MCA) gets the trace-route message from the transmission queue. The message is a trace-route message, therefore the MCA begins to record the activity information.
- The ACTIVREC attribute of the queue manager (QM1) is MSG, and the MQRO_ACTIVITY option is specified in the Report field of the message descriptor, therefore the MCA will later generate an activity report. The RecordedActivities parameter value in the TraceRoute PCF group is incremented by 1.
- The MCA checks that the MaxActivities value in the TraceRoute PCF group has not been exceeded.
- Before the message is forwarded to QM2 the MCA follows the algorithm that is described in Forwarding (steps 1 on page 996, 4 on page 996, and 5 on page 996) and the MCA chooses to send the message.
- The MCA then generates an activity report and puts it on the reply queue (ACTIV.REPLY.Q).

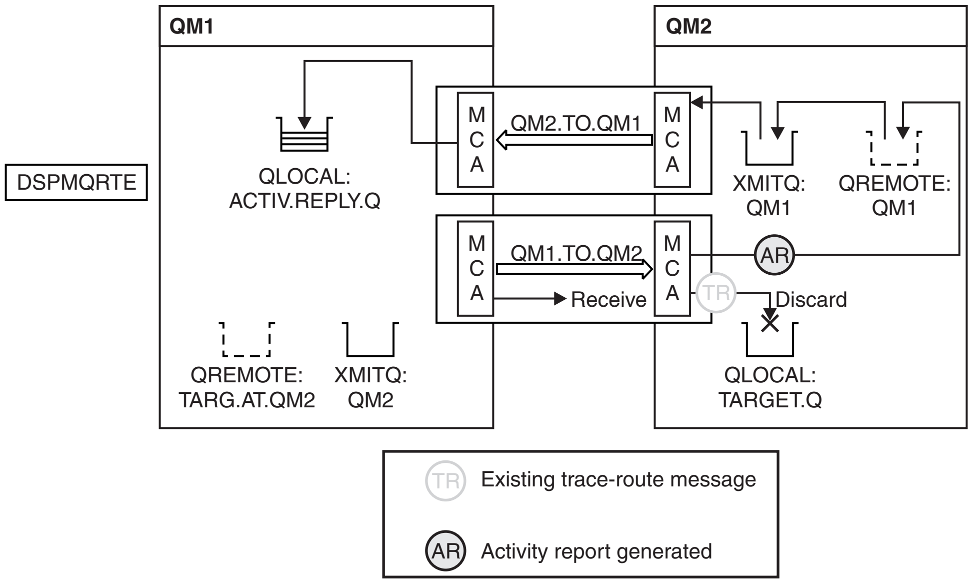


Figure 96. Requesting activity reports, Diagram 3

- The receiving MCA receives the trace-route message from the channel. The message is a trace-route message, therefore the MCA begins to record the information about the activity.
- If the queue manager that the trace-route message has come from is Version 5.3.1 or earlier, the MCA increments the DiscontinuityCount parameter of the TraceRoute PCF by 1. This is not the case here.
- The ACTIVREC attribute of the queue manager (QM2) is MSG, and the MQRO_ACTIVITY option is specified, therefore the MCA will generate an activity report. The RecordedActivities parameter value is incremented by 1.
- The target queue is a local queue, therefore the message is discarded with feedback MQFB_NOT_DELIVERED, in accordance with the Deliver parameter value in the TraceRoute PCF group.
- The MCA then generates the final activity report and puts it on the reply queue. This resolves to the transmission queue that is associated with queue manager QM1 and the activity report is returned to queue manager QM1 (ACTIV.REPLY.Q).

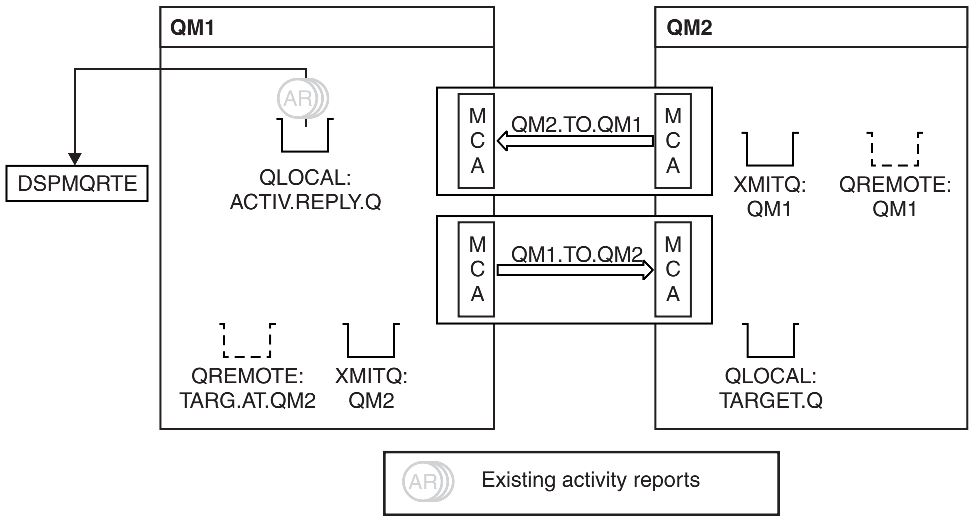


Figure 97. Requesting activity reports, Diagram 4

- Meanwhile, DSPMQRTE has been continually performing MQGETs on the reply queue (ACTIV.REPLY.Q), waiting for activity reports. It will wait for up to 120 seconds (60 seconds longer than the expiry time of the trace-route message) since -w was not specified when DSPMQRTE was started.
- DSPMQRTE gets the 3 activity reports off the reply queue.
- The activity reports are ordered using the RecordedActivities, UnrecordedActivities, and DiscontinuityCount parameters in the TraceRoute PCF group for each of the activities. The only value that is non-zero in this example is RecordedActivities, therefore this is the only parameter that is actually used.
- The program ends as soon as the discard operation is displayed. Even though the final operation was a discard, it is treated as though a put took place because the feedback is MQFB_NOT_DELIVERED.

  The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2
 -rq ACTIV.REPLY.Q'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2',
 queue manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
AMQ8666: Queue 'QM2' on queue manager 'QM1'.
AMQ8666: Queue 'TARGET.Q' on queue manager 'QM2'.
AMQ8652: DSPMQRTE command has finished.
```

Example 2 - Requesting a trace-route reply message:

Generate and deliver a trace-route message to the target queue

In this example the IBM MQ display route application connects to queue manager, QM1, and is used to generate and deliver a trace-route message to the target queue, TARGET.Q, on remote queue manager, QM2. The necessary option is specified so that activity information is accumulated in the trace-route message. On arrival at the target queue a trace-route reply message is requested, and the trace-route message is discarded.

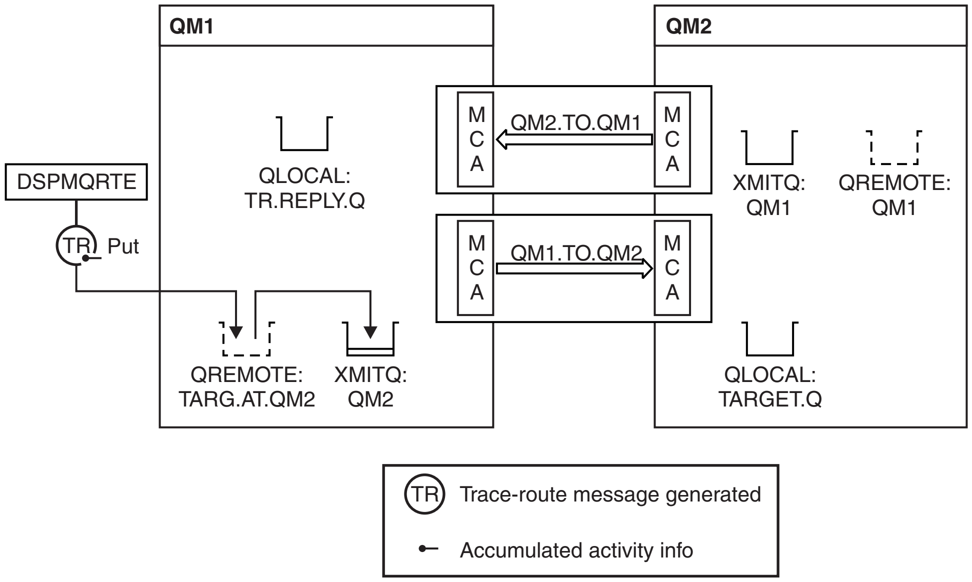


Figure 98. Requesting a trace-route reply message, Diagram 1

- The ROUTEREC attribute of each queue manager (QM1 and QM2) is set to MSG.
- The following command is issued:

  ```
  dspmqrte -m QM1 -q TARG.AT.QM2 -rq TR.REPLY.Q -ac -ar -ro discard
  ```

  QM1 is the name of the queue manager to which the IBM MQ display route application connects, TARG.AT.QM2 is the name of the target queue, and ACTIV.REPLY.Q is the name of the queue to which it is requested that all responses to the trace-route message are sent. The -ac option specifies that activity information is accumulated in the trace-route message, the -ar option specifies that all accumulated activity is sent to the reply-to queue that is specified by the -rq option (that is, TR.REPLY.Q). The -ro option specifies that report option MQRO_DISCARD_MSG is set which means that activity reports are not generated in this example.
- DSPMQRTE accumulates activity information in the trace-route message before the message is put on the target route. The queue manager attribute ROUTEREC must not be DISABLED for this to happen.

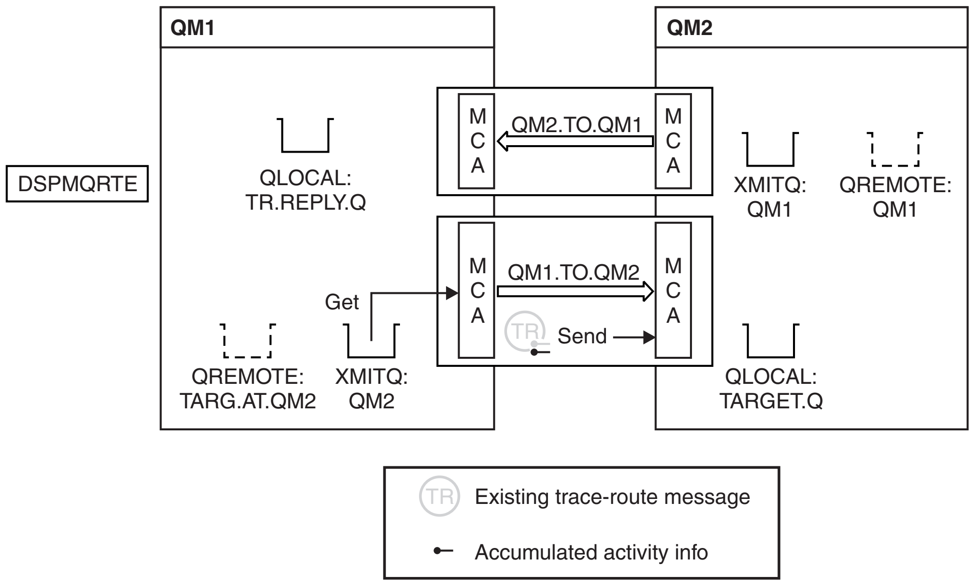


Figure 99. Requesting a trace-route reply message, Diagram 2

- The message is a trace-route message, therefore the sending MCA begins to record information about the activity.
- The queue manager attribute ROUTEREC on QM1 is not DISABLED, therefore the MCA accumulates the activity information within the message, before the message is forwarded to queue manager QM2.

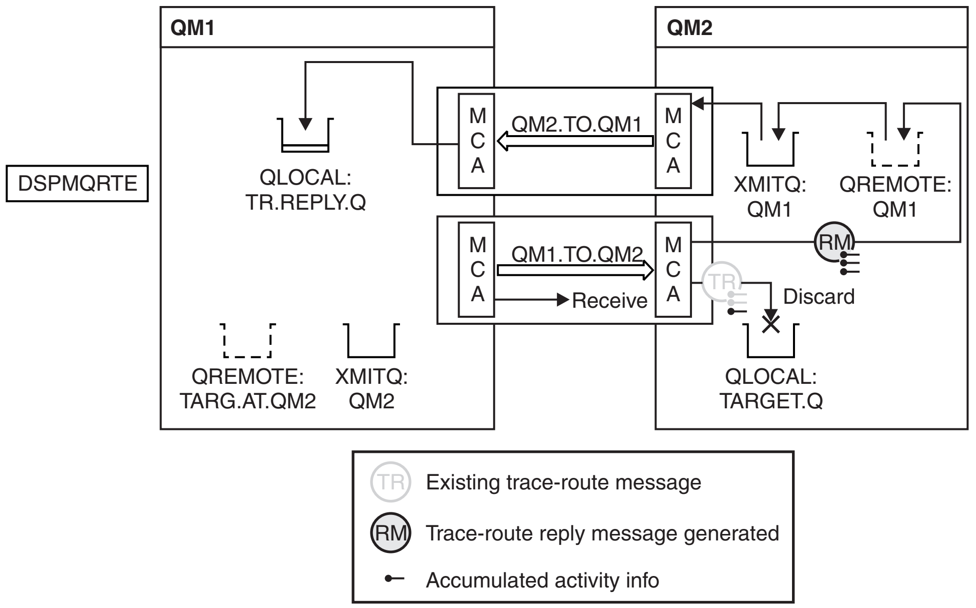


Figure 100. Requesting a trace-route reply message, Diagram 3

- The message is a trace-route message, therefore the receiving MCA begins to record information about the activity.

- The queue manager attribute ROUTEREC on QM2 is not DISABLED, therefore the MCA accumulates the information within the message.
- The target queue is a local queue, therefore the message is discarded with feedback MQFB_NOT_DELIVERED, in accordance with the Deliver parameter value in the TraceRoute PCF group.
- This is the last activity that will take place on the message, and because the queue manager attribute ROUTEREC on QM1 is not DISABLED, the MCA generates a trace-route reply message in accordance with the Accumulate value. The value of ROUTEREC is MSG, therefore the reply message is put on the reply queue. The reply message contains all the accumulated activity information from the trace-route message.

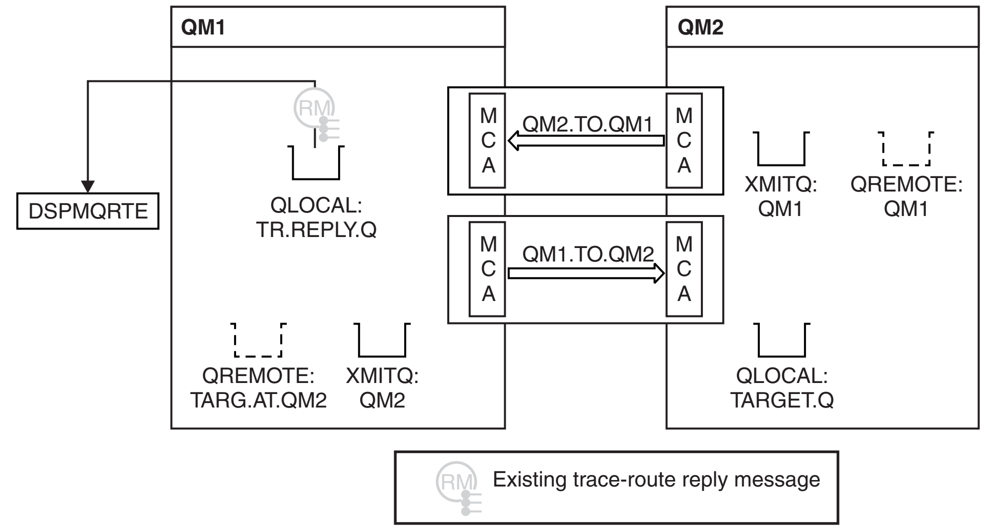


Figure 101. Requesting a trace-route reply message, Diagram 4

- Meanwhile DSPMQRTE is waiting for the trace-route reply message to return to the reply queue. When it returns, DSPMQRTE parses each activity that it contains and prints it out. The final operation is a discard operation. DSPMQRTE ends after it has been printed.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2 -rq
 TR.REPLY.Q'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2', queue
 manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
AMQ8666: Queue 'QM2' on queue manager 'QM1'.
AMQ8666: Queue 'TARGET.Q' on queue manager 'QM2'.
AMQ8652: DSPMQRTE command has finished.
```

Example 3 - Delivering activity reports to the system queue:

Detect when activity reports are delivered to queues other than the reply-to queue and use the IBM MQ display route application to read activity reports from the other queue.

This example is the same as "Example 1 - Requesting activity reports" on page 1010, except that QM2 now has the value of the ACTIVREC queue manage attribute set to QUEUE. Channel QM1.TO.QM2 must have been restarted for this to take effect.

This example demonstrates how to detect when activity reports are delivered to queues other than the reply-to queue. Once detected, the IBM MQ display route application is used to read activity reports from another queue.

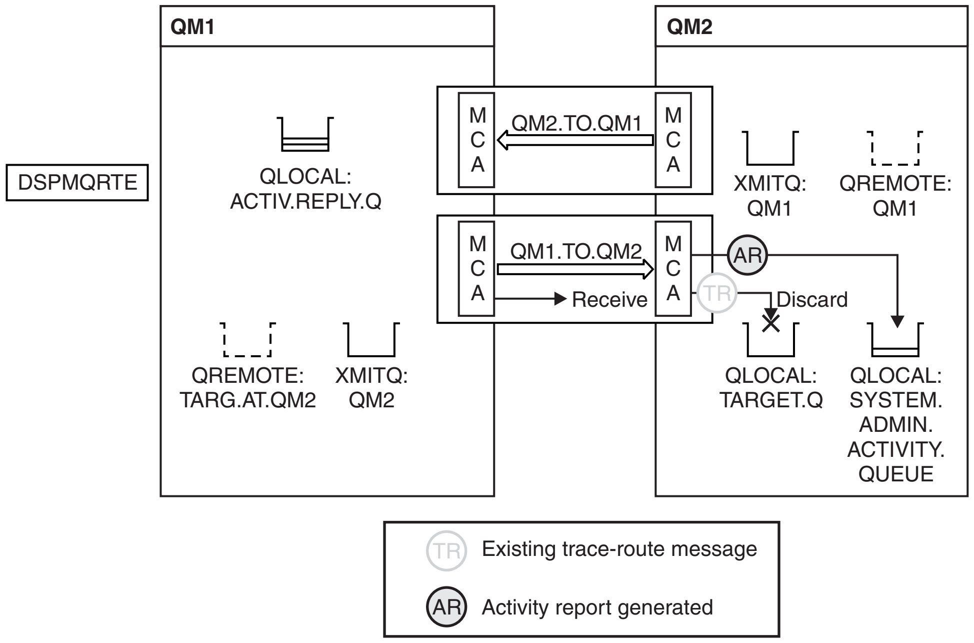


Figure 102. Delivering activity reports to the system queue, Diagram 1

- The message is a trace-route message, therefore the receiving MCA begins to record information about the activity.
- The value of the ACTIVREC queue manager attribute on QM2 is now QUEUE, therefore the MCA generates an activity report, but puts it on the system queue (SYSTEM.ADMIN.ACTIVITY.QUEUE) and not on the reply queue (ACTIV.REPLY.Q).

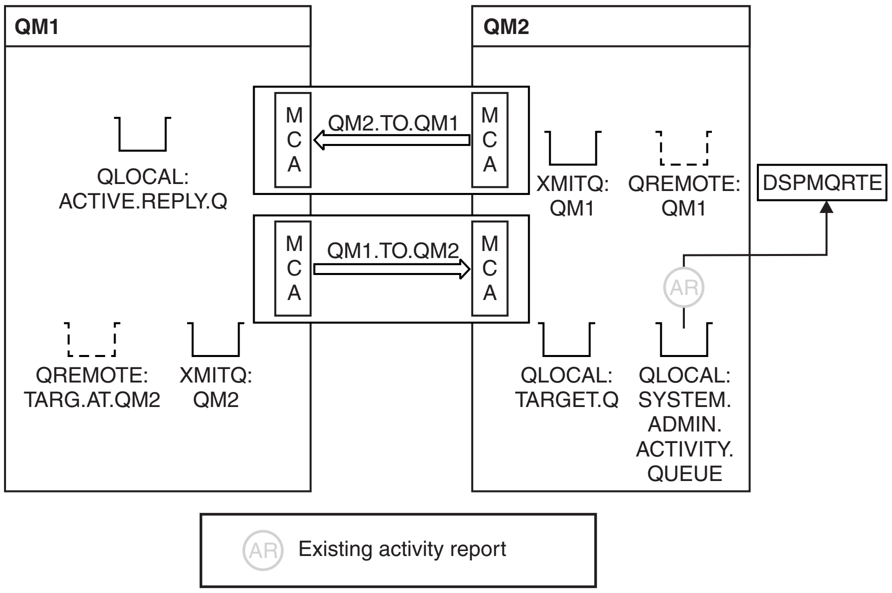


Figure 103. Delivering activity reports to the system queue, Diagram 2

- Meanwhile DSPMQRTE has been waiting for activity reports to arrive on ACTIV.REPLY.Q. Only two arrive. DSPMQRTE continues waiting for 120 seconds because it seems that the route is not yet complete.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2 -rq
         ACTIV.REPLY.Q -v outline identifiers'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2', queue
         manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\dspmqrte.exe'

 Operation:
  OperationType: Put

  Message:

   MQMD:
    MsgId: X'414D51204C41524745512020202020A3C9154220001502'
    CorrelId: X'414D51204C41524745512020202020A3C9154220001503'
  QMgrName: 'QM1                                            '
  QName: 'TARG.AT.QM2                                    '
  ResolvedQName: 'QM2                                            '
  RemoteQName: 'TARGET.Q                                       '
  RemoteQMgrName: 'QM2                                            '
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\runmqchl.EXE'

 Operation:
  OperationType: Get

  Message:

   MQMD:
    MsgId: X'414D51204C41524745512020202020A3C9154220001505'
    CorrelId: X'414D51204C41524745512020202020A3C9154220001502'

   EmbeddedMQMD:
    MsgId: X'414D51204C41524745512020202020A3C9154220001502'
    CorrelId: X'414D51204C41524745512020202020A3C9154220001503'
  QMgrName: 'QM1                                            '
  QName: 'QM2                                            '
  ResolvedQName: 'QM2                                            '

 Operation:
  OperationType: Send

  Message:

   MQMD:
    MsgId: X'414D51204C41524745512020202020A3C9154220001502'
    CorrelId: X'414D51204C41524745512020202020A3C9154220001503'
  QMgrName: 'QM1                                            '
  RemoteQMgrName: 'QM2                                            '
  ChannelName: 'QM1.TO.QM2          '
  ChannelType: Sender
  XmitQName: 'QM2                                            '
--------------------------------------------------------------------------------
AMQ8652: DSPMQRTE command has finished.
```

- The last operation that DSPMQRTE observed was a Send, therefore the channel is running. Now we must work out why we did not receive any more activity reports from queue manager QM2 (as identified in RemoteQMgrName).
- To check whether there is any activity information on the system queue, start DSPMQRTE on QM2 to try and collect more activity reports. Use the following command to start DSPMQRTE:

  ```
  dspmqrte -m QM2 -q SYSTEM.ADMIN.ACTIVITY.QUEUE
           -i 414D51204C41524745512020202020A3C9154220001502 -v outline
  ```

where 414D51204C41524745512020202020A3C9154220001502 is the MsgId of the trace-route message that was put.

- DSPMQRTE then performs a sequence of MQGETs again, waiting for responses on the system activity queue related to the trace-route message with the specified identifier.
- DSPMQRTE gets one more activity report, which it displays. DSPMQRTE determines that the preceding activity reports are missing, and displays a message saying this. We already know about this part of the route, however.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM2
         -q SYSTEM.ADMIN.ACTIVITY.QUEUE
         -i 414D51204C41524745512020202020A3C915420001502 -v outline'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------

Activity:
 Activity information unavailable.

--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\AMQRMPPA.EXE'

 Operation:
  OperationType: Receive
  QMgrName: 'QM2                                             '
  RemoteQMgrName: 'QM1                                           '
  ChannelName: 'QM1.TO.QM2          '
  ChannelType: Receiver

 Operation:
  OperationType: Discard
  QMgrName: 'QM2                                             '
  QName: 'TARGET.Q                                       '
  Feedback: NotDelivered

--------------------------------------------------------------------------------
AMQ8652: DSPMQRTE command has finished.
```

- This activity report indicates that the route information is now complete. No problem occurred.
- Just because route information is unavailable, or because DSPMQRTE cannot display all of the route, this does not mean that the message was not delivered. For example, the queue manager attributes of different queue managers might be different, or a reply queue might not be defined to get the response back.

Example 4 - Diagnosing a channel problem:

Diagnose a problem in which the trace-route message does not reach the target queue

In this example the IBM MQ display route application connects to queue manager, QM1, generates a trace-route message, then attempts to deliver it to the target queue, TARGET.Q, on remote queue manager, QM2. In this example the trace-route message does not reach the target queue. The available activity report is used to diagnose the problem.

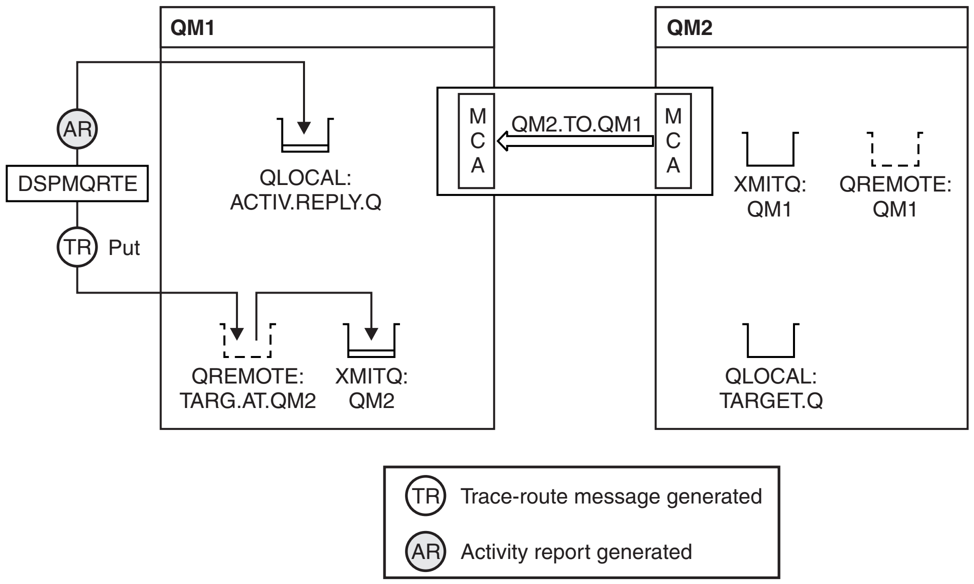


Figure 104. Diagnosing a channel problem

- In this example, the channel QM1.TO.QM2 is not running.
- DSPMQRTE puts a trace-route message (as in example 1) to the target queue and generates an activity report.
- There is no MCA to get the message from the transmission queue (QM2), therefore this is the only activity report that DSPMQRTE gets back from the reply queue. This time the fact that the route is not complete does indicate a problem. The administrator can use the transmission queue found in ResolvedQName to investigate why the transmission queue is not being serviced.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2
         -rq ACTIV.REPLY.Q -v outline'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2',
         queue manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\dspmqrte.exe'

 Operation:
  OperationType: Put
  QMgrName: 'QM1                                             '
  QName: 'TARG.AT.QM2                                     '
  ResolvedQName: 'QM2                                             '
  RemoteQName: 'TARGET.Q                                        '
  RemoteQMgrName: 'QM2                                             '

--------------------------------------------------------------------------------
 AMQ8652: DSPMQRTE command has finished.
```

Activity report reference

Use this page to obtain an overview of the activity report message format. The activity report message data contains the parameters that describe the activity.

Activity report format:

Activity reports are standard IBM MQ report messages containing a message descriptor and message data. Activity reports are PCF messages generated by applications that have performed an activity on behalf of a message as it has been routed through a queue manager network.

Activity reports contain the following information:

A message descriptor
An MQMD structure

Message data
Consists of the following:
- An embedded PCF header (MQEPH).
- Activity report message data.

Activity report message data consists of the *Activity* PCF group and, if generated for a trace-route message, the *TraceRoute* PCF group.

Table 119 on page 1023 shows the structure of these reports, including parameters that are returned only under certain conditions.

Table 119. Activity report format

MQMD structure	Embedded PCF header MQEPH structure	Activity report message data
Structure identifier	Structure identifier	Activity
Structure version	Structure version	Activity application name
Report options	Structure length	Activity application type
Message type	Encoding	Activity description
Expiration time	Coded character set ID	Operation
Feedback	Message format	Operation type
Encoding	Flags	Operation date
Coded character set ID	PCF header (MQCFH)	Operation time
Message format	Structure type	Message
Priority	Structure length	Message length
Persistence	Structure version	MQMD [8]
Message identifier	Command identifier	EmbeddedMQMD
Correlation identifier	Message sequence number	Queue manager name
Backout count	Control options	Queue sharing group name
Reply-to queue	Completion code	Queue name [1] [2] [3] [7]
Reply-to queue manager	Reason code	Resolved queue name [1] [3] [7]
User identifier	Parameter count	Remote queue name [3] [7]
Accounting token		Remote queue manager name [2] [3] [4] [5] [7]
Application identity data		Subscription level [9]
Application type		Subscription identifier [9]
Application name		Feedback [2] [10]
Put date		Channel name [4] [5]
Put time		Channel type [4] [5]
Application origin data		Transmission queue name [5]
Group identifier		TraceRoute [6]
Message sequence number		Detail
Offset		Recorded activities
Message flags		Unrecorded activities
Original length		Discontinuity count
		Max activities
		Accumulate
		Deliver

Notes:

1. Returned for Get and Browse operations.
2. Returned for Discard operations.
3. Returned for Put, Put Reply, and Put Report operations.
4. Returned for Receive operations.
5. Returned for Send operations.
6. Returned for trace-route messages.
7. Not returned for Put operations to a topic, contained within Publish activities.
8. Not returned for Excluded Publish operations. For Publish and Discarded Publish operations, returned containing a subset of parameters.
9. Returned for Publish, Discarded Publish, and Excluded Publish operations.
10. Returned for Discarded Publish and Excluded Publish operations.

Activity report MQMD (message descriptor):

Use this page to view the values contained by the MQMD structure for an activity report

StrucId
Structure identifier:

Data type
MQCHAR4

Value MQMD_STRUC_ID.

Version
Structure version number

Data type
MQLONG

Values
Copied from the original message descriptor. Possible values are:

MQMD_VERSION_1
Version-1 message descriptor structure, supported in all environments.

MQMD_VERSION_2
Version-2 message descriptor structure, supported on AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, and all IBM MQ MQI clients connected to these systems.

Report Options for further report messages

Data type
MQLONG

Value If MQRO_PASS_DISCARD_AND_EXPIRY or MQRO_DISCARD_MSG were specified in the *Report* field of the original message descriptor:

MQRO_DISCARD
The report is discarded if it cannot be delivered to the destination queue.

Otherwise:

MQRO_NONE
No reports required.

MsgType
Indicates type of message

Data type
MQLONG

Value MQMT_REPORT

Expiry Report message lifetime

Data type
MQLONG

Value If the *Report* field in the original message descriptor is specified as MQRO_PASS_DISCARD_AND_EXPIRY, the remaining expiry time from the original message is used.

Otherwise:

MQEI_UNLIMITED
The report does not have an expiry time.

Feedback

Description:	Feedback or reason code.
Data type:	MQLONG.
Value:	**MQFB_ACTIVITY** Activity report.

Encoding

Description:	Numeric encoding of report message data.
Data type:	MQLONG.
Value:	MQENC_NATIVE.

CodedCharSetId

Description:	Character set identifier of report message data.
Data type:	MQLONG.
Value:	Set as appropriate.

Format

Description:	Format name of report message data
Data type:	MQCHAR8.
Value:	**MQFMT_EMBEDDED_PCF** Embedded PCF message.

Priority

Description:	Report message priority.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

Persistence

Description:	Report message persistence.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

MsgId

Description:	Message identifier.
Data type:	MQBYTE24.
Values:	If the *Report* field in the original message descriptor is specified as MQRO_PASS_MSG_ID, the message identifier from the original message is used. Otherwise, a unique value will be generated by the queue manager.

CorrelId

Description:	Correlation identifier.
Data type:	MQBYTE24.
Value:	If the *Report* field in the original message descriptor is specified as MQRO_PASS_CORREL_ID, the correlation identifier from the original message is used. Otherwise, the message identifier is copied from the original message.

BackoutCount

Description:	Backout counter.
Data type:	MQLONG.
Value:	0.

ReplyToQ

Description:	Name of reply queue.
Data type:	MQCHAR48.
Values:	Blank.

ReplyToQMgr

Description:	Name of reply queue manager.
Data type:	MQCHAR48.
Value:	The queue manager name that generated the report message.

UserIdentifier

Description:	The user identifier of the application that generated the report message.
Data type:	MQCHAR12.
Value:	Copied from the original message descriptor.

AccountingToken

Description:	Accounting token that allows an application to charge for work done as a result of the message.
Data type:	MQBYTE32.
Value:	Copied from the original message descriptor.

ApplIdentityData

Description:	Application data relating to identity.
Data type:	MQCHAR32.
Values:	Copied from the original message descriptor.

PutApplType

Description:	Type of application that put the report message.
Data type:	MQLONG.
Value:	**MQAT_QMGR** Queue manager generated message.

PutApplName

Description:	Name of application that put the report message.
Data type:	MQCHAR28.
Value:	Either the first 28 bytes of the queue manager name, or the name of the MCA that generated the report message.

PutDate

Description:	Date when message was put.
Data type:	MQCHAR8.
Value:	As generated by the queue manager.

PutTime

Description:	Time when message was put.
Data type:	MQCHAR8.
Value:	As generated by the queue manager.

ApplOriginData

Description:	Application data relating to origin.
Data type:	MQCHAR4.
Value:	Blank.

If *Version* is MQMD_VERSION_2, the following additional fields are present:

GroupId

Description:	Identifies to which message group or logical message the physical message belongs.
Data type:	MQBYTE24.
Value:	Copied from the original message descriptor.

MsgSeqNumber

Description:	Sequence number of logical message within group.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

Offset

Description:	Offset of data in physical message from start of logical message.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

MsgFlags

Description:	Message flags that specify attributes of the message or control its processing.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

OriginalLength

Description:	Length of original message.
Data type:	MQLONG.
Value:	Copied from the original message descriptor.

Activity report MQEPH (Embedded PCF header):

Use this page to view the values contained by the MQEPH structure for an activity report

The MQEPH structure contains a description of both the PCF information that accompanies the message data of an activity report, and the application message data that follows it.

For an activity report, the MQEPH structure contains the following values:

StrucId

Description:	Structure identifier.
Data type:	MQCHAR4.
Value:	MQEPH_STRUC_ID.

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQEPH_VERSION_1.

StrucLength

Description:	Structure length.
Data type:	MQLONG.
Value:	Total length of the structure including the PCF parameter structures that follow it.

Encoding

Description:	Numeric encoding of the message data that follows the last PCF parameter structure.
Data type:	MQLONG.
Value:	If any data from the original application message data is included in the report message, the value will be copied from the *Encoding* field of the original message descriptor. Otherwise, 0.

CodedCharSetId

Description:	Character set identifier of the message data that follows the last PCF parameter structure.
Data type:	MQLONG.
Value:	If any data from the original application message data is included in the report message, the value will be copied from the *CodedCharSetId* field of the original message descriptor. Otherwise, MQCCSI_UNDEFINED.

Format

Description:	Format name of message data that follows the last PCF parameter structure.
Data type:	MQCHAR8.
Value:	If any data from the original application message data is included in the report message, the value will be copied from the *Format* field of the original message descriptor. Otherwise, MQFMT_NONE.

Flags

Description:	Flags that specify attributes of the structure or control its processing.
Data type:	MQLONG.
Value:	**MQEPH_CCSID_EMBEDDED** Specifies that the character set of the parameters containing character data is specified individually within the *CodedCharSetId* field in each structure.

PCFHeader

Description:	Programmable Command Format Header
Data type:	MQCFH.
Value:	See "Activity report MQCFH (PCF header)" on page 1030.

Activity report MQCFH (PCF header):

Use this page to view the PCF values contained by the MQCFH structure for an activity report

For an activity report, the MQCFH structure contains the following values:

Type

Description:	Structure type that identifies the content of the report message.
Data type:	MQLONG.
Value:	**MQCFT_REPORT** Message is a report.

StrucLength

Description:	Structure length.
Data type:	MQLONG.
Value:	**MQCFH_STRUC_LENGTH** Length in bytes of MQCFH structure.

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQCFH_VERSION_3

Command

Description:	Command identifier. This identifies the category of the message.
Data type:	MQLONG.
Values:	**MQCMD_ACTIVITY_MSG** Message activity.

MsgSeqNumber

Description:	Message sequence number. This is the sequence number of the message within a group of related messages.
Data type:	MQLONG.
Values:	1.

Control

Description:	Control options.
Data type:	MQLONG.
Values:	MQCFC_LAST.

CompCode

Description:	Completion code.
Data type:	MQLONG.
Values:	MQCC_OK.

Reason

Description:	Reason code qualifying completion code.
Data type:	MQLONG.
Values:	MQRC_NONE.

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Data type:	MQLONG.
Values:	1 or greater.

Activity report message data:

Use this page to view the parameters contained by the *Activity* PCF group in an activity report message. Some parameters are returned only when specific operations have been performed.

Activity report message data consists of the *Activity* PCF group and, if generated for a trace-route message, the *TraceRoute* PCF group. The *Activity* PCF group is detailed in this topic.

Some parameters, which are described as Operation-specific activity report message data, are returned only when specific operations have been performed.

For an activity report, the activity report message data contains the following parameters:

Activity

Description:	Grouped parameters describing the activity.
Identifier:	MQGACF_ACTIVITY.
Data type:	MQCFGR.
Included in PCF group:	None.
Parameters in PCF group:	*ActivityApplName* *ActivityApplType* *ActivityDescription* *Operation* *TraceRoute*
Returned:	Always.

ActivityApplName

Description:	Name of application that performed the activity.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Included in PCF group:	*Activity*.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ActivityApplType

Description:	Type of application that performed the activity.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*Activity*.
Returned:	Always.

ActivityDescription

Description:	Description of activity performed by the application.
Identifier:	MQCACF_ACTIVITY_DESCRIPTION.
Data type:	MQCFST.
Included in PCF group:	*Activity*.
Maximum length:	64
Returned:	Always.

Operation

Description:	Grouped parameters describing an operation of the activity.
Identifier:	MQGACF_OPERATION.
Data type:	MQCFGR.
Included in PCF group:	*Activity*.
Parameters in PCF group:	*OperationType* *OperationDate* *OperationTime* *Message* *QMgrName* *QSGName* **Note:** Additional parameters are returned in this group depending on the operation type. These additional parameters are described as Operation-specific activity report message data.
Returned:	One *Operation* PCF group per operation in the activity.

OperationType

Description:	Type of operation performed.
Identifier:	MQIACF_OPERATION_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Values:	MQOPER_*.
Returned:	Always.

OperationDate

Description:	Date when the operation was performed.
Identifier:	MQCACF_OPERATION_DATE.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

OperationTime

Description:	Time when the operation was performed.
Identifier:	MQCACF_OPERATION_TIME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_TIME_LENGTH.
Returned:	Always.

Message

Description:	Grouped parameters describing the message that caused the activity.
Identifier:	MQGACF_MESSAGE.
Data type:	MQCFGR.
Included in PCF group:	*Operation*.
Parameters in group:	*MsgLength* *MQMD* *EmbeddedMQMD*
Returned:	Always, except for Excluded Publish operations.

MsgLength

Description:	Length of the message that caused the activity, before the activity occurred.
Identifier:	MQIACF_MSG_LENGTH.
Data type:	MQCFIN.
Included in PCF group:	*Message*.
Returned:	Always.

MQMD

Description:	Grouped parameters related to the message descriptor of the message that caused the activity.
Identifier:	MQGACF_MQMD.
Data type:	MQCFGR.
Included in PCF group:	*Message*.
Parameters in group:	*StrucId* *Version* *Report* *MsgType* *Expiry* *Feedback* *Encoding* *CodedCharSetId* *Format* *Priority* *Persistence* *MsgId* *CorrelId* *BackoutCount* *ReplyToQ* *ReplyToQMgr* *UserIdentifier* *AccountingToken* *ApplIdentityData* *PutApplType* *PutApplName* *PutDate* *PutTime* *ApplOriginData* *GroupId* *MsgSeqNumber* *Offset* *MsgFlags* *OriginalLength*
Returned:	Always, except for Excluded Publish operations.

EmbeddedMQMD

Description:	Grouped parameters describing the message descriptor embedded within a message on a transmission queue.
Identifier:	MQGACF_EMBEDDDED_MQMD.
Data type:	MQCFGR.
Included in PCF group:	*Message*.

Parameters in group:	*StrucId* *Version* *Report* *MsgType* *Expiry* *Feedback* *Encoding* *CodedCharSetId* *Format* *Priority* *Persistence* *MsgId* *CorrelId* *BackoutCount* *ReplyToQ* *ReplyToQMgr* *UserIdentifier* *AccountingToken* *ApplIdentityData* *PutApplType* *PutApplName* *PutDate* *PutTime* *ApplOriginData* *GroupId* *MsgSeqNumber* *Offset* *MsgFlags* *OriginalLength*
Returned:	For Get operations where the queue resolves to a transmission queue.

StrucId

Description:	Structure identifier
Identifier:	MQCACF_STRUC_ID.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	4.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Version

Description:	Structure version number.
Identifier:	MQIACF_VERSION.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Report

Description:	Options for report messages.
Identifier:	MQIACF_REPORT.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

MsgType

Description:	Indicates type of message.
Identifier:	MQIACF_MSG_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Expiry

Description:	Message lifetime.
Identifier:	MQIACF_EXPIRY.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Feedback

Description:	Feedback or reason code.
Identifier:	MQIACF_FEEDBACK.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Encoding

Description:	Numeric encoding of message data.
Identifier:	MQIACF_ENCODING.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

CodedCharSetId

Description:	Character set identifier of message data.
Identifier:	MQIA_CODED_CHAR_SET_ID.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Format

Description:	Format name of message data
Identifier:	MQCACH_FORMAT_NAME.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_FORMAT_LENGTH.
Returned:	Always, except for Excluded Publish operations.

Priority

Description:	Message priority.
Identifier:	MQIACF_PRIORITY.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations.

Persistence

Description:	Message persistence.
Identifier:	MQIACF_PERSISTENCE.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations.

MsgId

Description:	Message identifier.
Identifier:	MQBACF_MSG_ID.
Data type:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_MSG_ID_LENGTH.
Returned:	Always, except for Excluded Publish operations.

CorrelId

Description:	Correlation identifier.
Identifier:	MQBACF_CORREL_ID.
Data type:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_CORREL_ID_LENGTH.
Returned:	Always, except for Excluded Publish operations.

BackoutCount

Description:	Backout counter.
Identifier:	MQIACF_BACKOUT_COUNT.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

ReplyToQ

Description:	Name of reply queue.
Identifier:	MQCACF_REPLY_TO_QUEUE.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish operations.

ReplyToQMgr

Description:	Name of reply queue manager.
Identifier:	MQCACF_REPLY_TO_Q_MGR.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

UserIdentifier

Description:	The user identifier of the application that originated the message.
Identifier:	MQCACF_USER_IDENTIFIER.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

AccountingToken

Description:	Accounting token that allows an application to charge for work done as a result of the message.
Identifier:	MQBACF_ACCOUNTING_TOKEN.
Data type:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_ACCOUNTING_TOKEN_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

ApplIdentityData

Description:	Application data relating to identity.
Identifier:	MQCACF_APPL_IDENTITY_DATA.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_IDENTITY_DATA_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

PutApplType

Description:	Type of application that put the message.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutApplName

Description:	Name of application that put the message.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutDate

Description:	Date when message was put.
Identifier:	MQCACF_PUT_DATE.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_PUT_DATE_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutTime

Description:	Time when message was put.
Identifier:	MQCACF_PUT_TIME.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_PUT_TIME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

ApplOriginData

Description:	Application data relating to origin.
Identifier:	MQCACF_APPL_ORIGIN_DATA.
Data type:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_ORIGIN_DATA_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

GroupId

Description:	Identifies to which message group or logical message the physical message belongs.
Identifier:	MQBACF_GROUP_ID.
Data type:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_GROUP_ID_LENGTH.
Returned:	If the *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

MsgSeqNumber

Description:	Sequence number of logical message within group.
Identifier:	MQIACH_MSG_SEQUENCE_NUMBER.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

Offset

Description:	Offset of data in physical message from start of logical message.
Identifier:	MQIACF_OFFSET.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

MsgFlags

Description:	Message flags that specify attributes of the message or control its processing.
Identifier:	MQIACF_MSG_FLAGS.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

OriginalLength

Description:	Length of original message.
Identifier:	MQIACF_ORIGINAL_LENGTH.
Data type:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

QMgrName

Description:	Name of the queue manager where the activity was performed.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

QSGName

Description:	Name of the queue-sharing group to which the queue manager where the activity was performed belongs.
Identifier:	MQCA_QSG_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_QSG_NAME_LENGTH
Returned:	If the activity was performed on an IBM MQ for z/OS queue manager.

TraceRoute

Description:	Grouped parameters specifying attributes of the trace-route message.
Identifier:	MQGACF_TRACE_ROUTE.
Data type:	MQCFGR.
Contained in PCF group:	*Activity*.
Parameters in group:	*Detail* *RecordedActivities* *UnrecordedActivities* *DiscontinuityCount* *MaxActivities* *Accumulate* *Forward* *Deliver*
Returned:	If the activity was performed on behalf of the trace-route message.

The values of the parameters in the *TraceRoute* PCF group are those from the trace-route message at the time the activity report was generated.

Operation-specific activity report message data:

Use this page to view the additional PCF parameters that might be returned in the PCF group *Operation* in an activity report, depending on the value of the *OperationType* parameter

The additional parameters vary depending on the following operation types:

Get/Browse (MQOPER_GET/MQOPER_BROWSE):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Get/Browse (MQOPER_GET/MQOPER_BROWSE) operation type (a message on a queue was got, or browsed).

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always.

ResolvedQName

Description:	The name that the opened queue resolves to.
Identifier:	MQCACF_RESOLVED_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always.

Discard (MQOPER_DISCARD):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Discard (MQOPER_DISCARD) operation type (a message was discarded).

Feedback

Description:	The reason for the message being discarded.
Identifier:	MQIACF_FEEDBACK.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH
Included in PCF group:	*Operation*.
Returned:	If the message was discarded because it was unsuccessfully put to a queue.

RemoteQMgrName

Description:	The name of the queue manager to which the message was destined.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Included in PCF group:	*Operation*.
Returned:	If the value of *Feedback* is MQFB_NOT_FORWARDED.

Publish/Discarded Publish/Excluded Publish (MQOPER_PUBLISH/MQOPER_DISCARDED_PUBLISH/ MQOPER_EXCLUDED_PUBLISH):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Publish/Discarded Publish/Excluded Publish (MQOPER_PUBLISH/ MQOPER_DISCARDED_PUBLISH/MQOPER_EXCLUDED_PUBLISH) operation type (a publish/subscribe message was delivered, discarded, or excluded).

SubId

Description:	The subscription identifier.
Identifier:	MQBACF_SUB_ID.
Data type:	MQCFBS.
Included in PCF group:	*Operation*.
Returned:	Always.

SubLevel

Description:	The subscription level.
Identifier:	MQIACF_SUB_LEVEL.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

Feedback

Description:	The reason for discarding the message.
Identifier:	MQIACF_FEEDBACK.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	If the message was discarded because it was not delivered to a subscriber, or the message was not delivered because the subscriber was excluded.

The Publish operation MQOPER_PUBLISH provides information about a message delivered to a particular subscriber. This operation describes the elements of the onward message that might have changed from the message described in the associated Put operation. Similarly to a Put operation, it contains a message group MQGACF_MESSAGE and, inside that, an MQMD group MQGACF_MQMD. However, this MQMD group contains only the following fields, which can be overridden by a subscriber: *Format, Priority, Persistence, MsgId, CorrelId, UserIdentifier, AccountingToken, ApplIdentityData.*

The *SubId* and *SubLevel* of the subscriber are included in the operation information. You can use the *SubID* with the MQCMD_INQUIRE_SUBSCRIBER PCF command to retrieve all other attributes for a subscriber.

The Discarded Publish operation MQOPER_DISCARDED_PUBLISH is analogous to the Discard operation that is used when a message is not delivered in point-to-point messaging. A message is not delivered to a subscriber if the message was explicitly requested not to be delivered to a local destination and this subscriber specifies a local destination. A message is also considered not delivered if there is a problem getting the message to the destination queue, for example, because the queue is full.

The information in a Discarded Publish operation is the same as for a Publish operation, with the addition of a *Feedback* field that gives the reasons why the message was not delivered. This feedback field contains MQFB_* or MQRC_* values that are common with the MQOPER_DISCARD operation. The reason for discarding a publish, as opposed to excluding it, are the same as the reasons for discarding a put.

The Excluded Publish operation MQOPER_EXCLUDED_PUBLISH provides information about a subscriber that was considered for delivery of the message, because the topic on which the subscriber is subscribing matches that of the associated Put operation, but the message was not delivered to the subscriber because other selection criteria do not match with the message that is being put to the topic. As with a Discarded Publish operation, the *Feedback* field provides information about the reason why this subscription was excluded. However, unlike the Discarded Publish operation, no message-related information is provided because no message was generated for this subscriber.

Put/Put Reply/Put Report (MQOPER_PUT/MQOPER_PUT_REPLY/MQOPER_PUT_REPORT):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Put/Put Reply/Put Report (MQOPER_PUT/MQOPER_PUT_REPLY/MQOPER_PUT_REPORT) operation type (a message, reply message, or report message was put to a queue).

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always, apart from one exception: not returned if the Put operation is to a topic, contained within a publish activity.

ResolvedQName

Description:	The name that the opened queue resolves to.
Identifier:	MQCACF_RESOLVED_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	When the opened queue could be resolved. Not returned if the Put operation is to a topic, contained within a publish activity.

RemoteQName

Description:	The name of the opened queue, as it is known on the remote queue manager.
Identifier:	MQCA_REMOTE_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	If the opened queue is a remote queue. Not returned if the Put operation is to a topic, contained within a publish activity.

RemoteQMgrName

Description:	The name of the remote queue manager on which the remote queue is defined.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	If the opened queue is a remote queue. Not returned if the Put operation is to a topic, contained within a publish activity.

TopicString

Description:	The full topic string to which the message is being put.
Identifier:	MQCA_TOPIC_STRING.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Returned:	If the Put operation is to a topic, contained within a publish activity.

Feedback

Description:	The reason for the message being put on the dead-letter queue.
Identifier:	MQIACF_FEEDBACK.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	If the message was put on the dead-letter queue.

Receive (MQOPER_RECEIVE):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Receive (MQOPER_RECEIVE) operation type (a message was received on a channel).

ChannelName

Description:	The name of the channel on which the message was received.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	Always.

ChannelType

Description:	The type of channel on which the message was received.
Identifier:	MQIACH_CHANNEL_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

RemoteQMgrName

Description:	The name of the queue manager from which the message was received.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

Send (MQOPER_SEND):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Send (MQOPER_SEND) operation type (a message was sent on a channel).

ChannelName

Description:	The name of the channel where the message was sent.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

ChannelType

Description:	The type of channel where the message was sent.
Identifier:	MQIACH_CHANNEL_TYPE.
Data type:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

XmitQName

Description:	The transmission queue from which the message was retrieved.
Identifier:	MQCACH_XMIT_Q_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

RemoteQMgrName

Description:	The name of the remote queue manager to which the message was sent.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

Trace-route message reference

Use this page to obtain an overview of the trace-route message format. The trace-route message data includes parameters that describe the activities that the trace-route message has caused

Trace-route message format:

Trace-route messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the activities performed on a trace-route message as it has been routed through a queue manager network.

Trace-route messages contain the following information:

A message descriptor
An MQMD structure, with the *Format* field set to MQFMT_ADMIN or MQFMT_EMBEDDED_PCF.

Message data
Consists of either:

- A PCF header (MQCFH) and trace-route message data, if *Format* is set to MQFMT_ADMIN, or
- An embedded PCF header (MQEPH), trace-route message data, and additional user-specified message data, if *Format* is set to MQFMT_EMBEDDED_PCF.

When using the IBM MQ display route application to generate a trace-route message, *Format* is set to MQFMT_ADMIN.

The content of the trace-route message data is determined by the *Accumulate* parameter from the *TraceRoute* PCF group, as follows:

- If *Accumulate* is set to MQROUTE_ACCUMULATE_NONE, the trace-route message data contains the *TraceRoute* PCF group.
- If *Accumulate* is set to either MQROUTE_ACCUMULATE_IN_MSG or MQROUTE_ACCUMULATE_AND_REPLY, the trace-route message data contains the *TraceRoute* PCF group and zero or more *Activity* PCF groups.

Table 120 on page 1049 shows the structure of a trace-route message.

Table 120. Trace-route message format

MQMD structure	Embedded PCF header MQEPH structure	Trace-route message data
Structure identifier	Structure identifier	TraceRoute
Structure version	Structure version	Detail
Report options	Structure length	Recorded activities
Message type	Encoding	Unrecorded activities
Expiration time	Coded character set ID	Discontinuity count
Feedback	Message format	Max activities
Encoding	Flags	Accumulate
Coded character set ID	PCF header (MQCFH)	Deliver
Message format	Structure type	
Priority	Structure length	
Persistence	Structure version	
Message identifier	Command identifier	
Correlation identifier	Message sequence number	
Backout count	Control options	
Reply-to queue	Completion code	
Reply-to queue manager	Reason code	
User identifier	Parameter count	
Accounting token		
Application identity data		
Application type		
Application name		
Put date		
Put time		
Application origin data		
Group identifier		
Message sequence number		
Offset		
Message flags		
Original length		

Trace-route message MQMD (message descriptor):

Use this page to view the values contained by the MQMD structure for a trace-route message

StrucId

Description:	Structure identifier.
Data type:	MQCHAR4.
Value:	MQMD_STRUC_ID.

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	**MQMD_VERSION_1.**

Report

Description:	Options for report messages.
Data type:	MQLONG.
Value:	Set according to requirements. Common report options follow:

MQRO_DISCARD_MSG
The message is discarded on arrival to a local queue.

MQRO_PASS_DISCARD_AND_EXPIRY
Every response (activity reports or trace-route reply message) will have the report option MQRO_DISCARD_MSG set, and the remaining expiry passed on. This ensures that responses do not remain in the queue manager network indefinitely.

MsgType

Description:	Type of message.
Data type:	MQLONG.
Value:	If the *Accumulate* parameter in the TraceRoute group is specified as MQROUTE_ACCUMULATE_AND_REPLY, then message type is MQMT_REQUEST

Otherwise:

MQMT_DATAGRAM.

Expiry

Description:	Message lifetime.
Data type:	MQLONG.
Value:	Set according to requirements. This parameter can be used to ensure trace-route messages are not left in a queue manager network indefinitely.

Feedback

Description:	Feedback or reason code.
Data type:	MQLONG.
Value:	**MQFB_NONE.**

Encoding

Description:	Numeric encoding of message data.
Data type:	MQLONG.
Value:	Set as appropriate.

CodedCharSetId

Description:	Character set identifier of message data.
Data type:	MQLONG.
Value:	Set as appropriate.

Format

Description:	Format name of message data
Data type:	MQCHAR8.
Value:	**MQFMT_ADMIN** Admin message. No user data follows the *TraceRoute* PCF group. **MQFMT_EMBEDDED_PCF** Embedded PCF message. User data follows the *TraceRoute* PCF group.

Priority

Description:	Message priority.
Data type:	MQLONG.
Value:	Set according to requirements.

Persistence

Description:	Message persistence.
Data type:	MQLONG.
Value:	Set according to requirements.

MsgId

Description:	Message identifier.
Data type:	MQBYTE24.
Value:	Set according to requirements.

CorrelId

Description:	Correlation identifier.
Data type:	MQBYTE24.
Value:	Set according to requirements.

BackoutCount

Description:	Backout counter.
Data type:	MQLONG.
Value:	0.

ReplyToQ

Description:	Name of reply queue.
Data type:	MQCHAR48.
Values:	Set according to requirements. If *MsgType* is set to MQMT_REQUEST or if *Report* has any report generating options set, then this parameter must be non-blank.

ReplyToQMgr

Description:	Name of reply queue manager.
Data type:	MQCHAR48.
Value:	Set according to requirements.

UserIdentifier

Description:	The user identifier of the application that originated the message.
Data type:	MQCHAR12.
Value:	Set as normal.

AccountingToken

Description:	Accounting token that allows an application to charge for work done as a result of the message.
Data type:	MQBYTE32.
Value:	Set as normal.

ApplIdentityData

Description:	Application data relating to identity.
Data type:	MQCHAR32.
Values:	Set as normal.

PutApplType

Description:	Type of application that put the message.
Data type:	MQLONG.
Value:	Set as normal.

PutApplName

Description:	Name of application that put the message.
Data type:	MQCHAR28.
Value:	Set as normal.

PutDate

Description:	Date when message was put.
Data type:	MQCHAR8.
Value:	Set as normal.

PutTime

Description:	Time when message was put.
Data type:	MQCHAR8.
Value:	Set as normal.

ApplOriginData

Description:	Application data relating to origin.
Data type:	MQCHAR4.
Value:	Set as normal..

Trace-route message MQEPH (Embedded PCF header):

Use this page to view the values contained by the MQEPH structure for a trace-route message

The MQEPH structure contains a description of both the PCF information that accompanies the message data of a trace-route message, and the application message data that follows it. An MQEPH structure is used only if additional user message data follows the TraceRoute PCF group.

For a trace-route message, the MQEPH structure contains the following values:

StrucId

Description:	Structure identifier.
Data type:	MQCHAR4.
Value:	MQEPH_STRUC_ID.

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQEPH_VERSION_1.

StrucLength

Description:	Structure length.
Data type:	MQLONG.
Value:	Total length of the structure including the PCF parameter structures that follow it.

Encoding

Description:	Numeric encoding of the message data that follows the last PCF parameter structure.
Data type:	MQLONG.
Value:	The encoding of the message data.

CodedCharSetId

Description:	Character set identifier of the message data that follows the last PCF parameter structure.
Data type:	MQLONG.
Value:	The character set of the message data.

Format

Description:	Format name of the message data that follows the last PCF parameter structure.
Data type:	MQCHAR8.
Value:	The format name of the message data.

Flags

Description:	Flags that specify attributes of the structure or control its processing.
Data type:	MQLONG.
Value:	**MQEPH_NONE** No flags specified. **MQEPH_CCSID_EMBEDDED** Specifies that the character set of the parameters containing character data is specified individually within the *CodedCharSetId* field in each structure.

PCFHeader

Description:	Programmable Command Format Header
Data type:	MQCFH.
Value:	See "Trace-route message MQCFH (PCF header)."

Trace-route message MQCFH (PCF header):

Use this page to view the PCF values contained by the MQCFH structure for a trace-route message

For a trace-route message, the MQCFH structure contains the following values:

Type

Description:	Structure type that identifies the content of the message.
Data type:	MQLONG.
Value:	**MQCFT_TRACE_ROUTE** Message is a trace-route message.

StrucLength

Description:	Structure length.
Data type:	MQLONG.
Value:	**MQCFH_STRUC_LENGTH** Length in bytes of MQCFH structure.

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQCFH_VERSION_3

Command

Description:	Command identifier. This identifies the category of the message.
Data type:	MQLONG.
Values:	**MQCMD_TRACE_ROUTE** Trace-route message.

MsgSeqNumber

Description:	Message sequence number. This is the sequence number of the message within a group of related messages.
Data type:	MQLONG.
Values:	1.

Control

Description:	Control options.
Data type:	MQLONG.
Values:	MQCFC_LAST.

CompCode

Description:	Completion code.
Data type:	MQLONG.
Values:	MQCC_OK.

Reason

Description:	Reason code qualifying completion code.
Data type:	MQLONG.
Values:	MQRC_NONE.

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Data type:	MQLONG.
Values:	1 or greater.

Trace-route message data:

Use this page to view the parameters that make up the *TraceRoute* PCF group part of trace-route message data

The content of trace-route message data depends on the *Accumulate* parameter from the *TraceRoute* PCF group. Trace-route message data consists of the *TraceRoute* PCF group, and zero or more *Activity* PCF groups. The *TraceRoute* PCF group is detailed in this topic. Refer to the related information for details of the *Activity* PCF group.

Trace-route message data contains the following parameters:

TraceRoute

Description:	Grouped parameters specifying attributes of the trace-route message. For a trace-route message, some of these parameters can be altered to control how it is processed.
Identifier:	MQGACF_TRACE_ROUTE.
Data type:	MQCFGR.
Contained in PCF group:	None.
Parameters in group:	*Detail* *RecordedActivities* *UnrecordedActivities* *DiscontinuityCount* *MaxActivities* *Accumulate* *Forward* *Deliver*

Detail

Description:	The detail level that will be recorded for the activity.
Identifier:	MQIACF_ROUTE_DETAIL.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Values:	**MQROUTE_DETAIL_LOW** Activities performed by user-written application are recorded. **MQROUTE_DETAIL_MEDIUM** Activities specified in MQROUTE_DETAIL_LOW are recorded. Additionally, activities performed by MCAs are recorded. **MQROUTE_DETAIL_HIGH** Activities specified in MQROUTE_DETAIL_LOW, and MQROUTE_DETAIL_MEDIUM are recorded. MCAs do not record any further activity information at this level of detail. This option is only available to user-written applications that are to record further activity information.

RecordedActivities

Description:	The number of activities that the trace-route message has caused, where information was recorded.
Identifier:	MQIACF_RECORDED_ACTIVITIES.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

UnrecordedActivities

Description:	The number of activities that the trace-route message has caused, where information was not recorded.
Identifier:	MQIACF_UNRECORDED_ACTIVITIES.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

DiscontinuityCount

Description:	The number of times a trace-route message has been received from a queue manager that does not support trace-route messaging.
Identifier:	MQIACF_DISCONTINUITY_COUNT.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

MaxActivities

Description:	The maximum number of activities the trace-route message can be involved in before it stops being processed.
Identifier:	MQIACF_MAX_ACTIVITIES.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Value:	**A positive integer** The maximum number of activities. **MQROUTE_UNLIMITED_ACTIVITIES** An unlimited number of activities.

Accumulate

Description:	Specifies whether activity information is accumulated within the trace-route message, and whether a reply message containing the accumulated activity information is generated before the trace-route message is discarded or is put on a non-transmission queue.
Identifier:	MQIACF_ROUTE_ACCUMULATION.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

Value:

MQROUTE_ACCUMULATE_NONE
Activity information is not accumulated in the message data of the trace-route message.

MQROUTE_ACCUMULATE_IN_MSG
Activity information is accumulated in the message data of the trace-route message.

MQROUTE_ACCUMULATE_AND_REPLY
Activity information is accumulated in the message data of the trace-route message, and a trace-route reply message will be generated.

Forward

Description:	Specifies queue managers that the trace-route message can be forwarded to. When determining whether to forward a message to a remote queue manager, queue managers use the algorithm that is described in Forwarding.
Identifier:	MQIACF_ROUTE_FORWARDING.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

Value:

MQROUTE_FORWARD_IF_SUPPORTED
The trace-route message is only forwarded to queue managers that will honor the value of the *Deliver* parameter from the *TraceRoute* group.

MQROUTE_FORWARD_ALL
The trace-route message is forwarded to any queue manager, regardless of whether the value of the *Deliver* parameter will be honored.

Deliver

Description:	Specifies the action to be taken if the trace-route message arrives at the destination queue successfully.
Identifier:	MQIACF_ROUTE_DELIVERY.
Data type:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

Value:

MQROUTE_DELIVER_YES
On arrival, the trace-route message is put on the target queue. Any application performing a destructive get on the target queue can receive the trace-route message.

MQROUTE_DELIVER_NO
On arrival, the trace-route message is discarded.

Trace-route reply message reference

Use this page to obtain an overview of the trace-route reply message format. The trace-route reply message data is a duplicate of the trace-route message data from the trace-route message for which it was generated

Trace-route reply message format:

Trace-route reply messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the activities performed on a trace-route message as it has been routed through a queue manager network.

Trace-route reply messages contain the following information:

A message descriptor
An MQMD structure

Message data
A PCF header (MQCFH) and trace-route reply message data

Trace-route reply message data consists of one or more *Activity* PCF groups.

When a trace-route message reaches its target queue, a trace-route reply message can be generated that contains a copy of the activity information from the trace-route message. The trace-route reply message will be delivered to a reply-to queue or to a system queue.

Table 121 on page 1060 shows the structure of a trace-route reply message, including parameters that are only returned under certain conditions.

Table 121. Trace-route reply message format

MQMD structure	**PCF header MQCFH structure**	**Trace-route reply message data**
Structure identifier	PCF header (MQCFH)	Activity
Structure version	Structure type	Activity application name
Report options	Structure length	Activity application type
Message type	Structure version	Activity description
Expiration time	Command identifier	Operation
Feedback	Message sequence number	Operation type
Encoding	Control options	Operation date
Coded character set ID	Completion code	Operation time
Message format	Reason code	Message
Priority	Parameter count	Message length
Persistence		MQMD
Message identifier		EmbeddedMQMD
Correlation identifier		Queue manager name
Backout count		Queue sharing group name
Reply-to queue		Queue name [1] [2] [3]
Reply-to queue manager		Resolved queue name [1] [3]
User identifier		Remote queue name [3]
Accounting token		Remote queue manager-name [2] [3] [4] [5]
Application identity data		Feedback [2]
Application type		Channel name [4] [5]
Application name		Channel type [4] [5]
Put date		Transmission queue name [5]
Put time		TraceRoute
Application origin data		Detail
Group identifier		Recorded activities
Message sequence number		Unrecorded activities
Offset		Discontinuity count
Message flags		Max activities
Original length		Accumulate
		Deliver

Note:

1. Returned for Get and Browse operations.
2. Returned for Discard operations.
3. Returned for Put, Put Reply, and Put Report operations.
4. Returned for Receive operations.
5. Returned for Send operations.

Trace-route reply message MQMD (message descriptor):

Use this page to view the values contained by the MQMD structure for a trace-route reply message

For a trace-route reply message, the MQMD structure contains the parameters described in Activity report message descriptor. Some of the parameter values in a trace-route reply message descriptor are different from those in an activity report message descriptor, as follows:

MsgType

Description:	Type of message.
Data type:	MQLONG.
Value:	**MQMT_REPLY**

Feedback

Description:	Feedback or reason code.
Data type:	MQLONG.
Value:	**MQFB_NONE**

Encoding

Description:	Numeric encoding of message data.
Data type:	MQLONG.
Value:	Copied from trace-route message descriptor.

CodedCharSetId

Description:	Character set identifier of message data.
Data type:	MQLONG.
Value:	Copied from trace-route message descriptor.

Format

Description:	Format name of message data
Data type:	MQCHAR8.
Value:	**MQFMT_ADMIN** Admin message.

Trace-route reply message MQCFH (PCF header):

Use this page to view the PCF values contained by the MQCFH structure for a trace-route reply message

The PCF header (MQCFH) for a trace-route reply message is the same as for a trace-route message.

Trace-route reply message data:

The trace-route reply message data is a duplicate of the trace-route message data from the trace-route message for which it was generated

The trace-route reply message data contains one or more *Activity* groups. The parameters are described in "Activity report message data" on page 1031.

Accounting and statistics messages

Queue managers generate accounting and statistics messages to record information about the MQI operations performed by IBM MQ applications, or to record information about the activities occurring in an IBM MQ system.

Accounting messages
Accounting messages are used to record information about the MQI operations performed by IBM MQ applications, see "Accounting messages" on page 1063.

Statistics messages
Statistics messages are used to record information about the activities occurring in an IBM MQ system, see "Statistics messages" on page 1066.

z/OS Accounting messages and statistics messages as described here are not available on IBM MQ for z/OS, but equivalent functionality is available through the System Management Facility (SMF).

Accounting and statistics messages are delivered to one of two system queues. User applications can retrieve the messages from these system queues and use the recorded information for various purposes:

- Account for application resource use.
- Record application activity.
- Capacity planning.
- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm that your queue manager network is running correctly.

Related concepts:

z/OS "Using System Management Facility" on page 1224
You can use SMF to collect statistics and accounting information. To use SMF, certain parameters must be set in z/OS and in IBM MQ.

Accounting messages

Accounting messages record information about the MQI operations performed by IBM MQ applications. An accounting message is a PCF message that contains a number of PCF structures.

When an application disconnects from a queue manager, an accounting message is generated and delivered to the system accounting queue (SYSTEM.ADMIN.ACCOUNTING.QUEUE). For long running IBM MQ applications, intermediate accounting messages are generated as follows:

- When the time since the connection was established exceeds the configured interval.
- When the time since the last intermediate accounting message exceeds the configured interval.

Accounting messages are in the following categories:

MQI accounting messages
MQI accounting messages contain information relating to the number of MQI calls made using a connection to a queue manager.

Queue accounting messages
Queue accounting messages contain information relating to the number of MQI calls made using connections to a queue manager, grouped by queue.

Each queue accounting message can contain up to 100 records, with every record relating to an activity performed by the application with respect to a specific queue.

Accounting messages are recorded only for local queues. If an application makes an MQI call against an alias queue, the accounting data is recorded against the base queue, and, for a remote queue, the accounting data is recorded against the transmission queue.

Related reference:

"MQI accounting message data" on page 1080
Use this page to view the structure of an MQI accounting message

"Queue accounting message data" on page 1091
Use this page to view the structure of a queue accounting message

Accounting message format:

Accounting messages comprise a set of PCF fields that consist of a message descriptor and message data.

Message descriptor

- An accounting message MQMD (message descriptor)

Accounting message data

- An accounting message MQCFH (PCF header)
- Accounting message data that is always returned
- Accounting message data that is returned if available

The accounting message MQCFH (PCF header) contains information about the application, and the interval for which the accounting data was recorded.

Accounting message data comprises PCF parameters that store the accounting information. The content of accounting messages depends on the message category as follows:

MQI accounting message
MQI accounting message data consists of a number of PCF parameters, but no PCF groups.

Queue accounting message

Queue accounting message data consists of a number of PCF parameters, and in the range 1 through 100 *QAccountingData* PCF groups.

There is one *QAccountingData* PCF group for every queue that had accounting data collected. If an application accesses more than 100 queues, multiple accounting messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as `MQCFC_LAST`.

Accounting information collection:

Use queue and queue manager attributes to control the collection of accounting information. You can also use MQCONNX options to control collection at the connection level.

MQI accounting information:

Use the queue manager attribute ACCTMQI to control the collection of MQI accounting information

To change the value of this attribute, use the MQSC command, `ALTER QMGR`, and specify the parameter `ACCTMQI`. Accounting messages are generated only for connections that begin after accounting is enabled. The `ACCTMQI` parameter can have the following values:

ON MQI accounting information is collected for every connection to the queue manager.

OFF MQI accounting information is not collected. This is the default value.

For example, to enable MQI accounting information collection use the following MQSC command:

```
ALTER QMGR ACCTMQI(ON)
```

Queue accounting information:

Use the queue attribute ACCTQ and the queue manager attribute ACCTQ to control the collection of queue accounting information.

To change the value of the queue attribute, use the MQSC command, `ALTER QLOCAL` and specify the parameter `ACCTQ`. Accounting messages are generated only for connections that begin after accounting is enabled. The queue attribute ACCTQ can have the following values:

ON Queue accounting information for this queue is collected for every connection to the queue manager that opens the queue.

OFF Queue accounting information for this queue is not collected.

QMGR
The collection of queue accounting information for this queue is controlled according to the value of the queue manager attribute ACCTQ. This is the default value.

To change the value of the queue manager attribute, use the MQSC command, `ALTER QMGR` and specify the parameter `ACCTQ`. The queue manager attribute ACCTQ can have the following values:

ON Queue accounting information is collected for queues that have the queue attribute ACCTQ set as QMGR.

OFF Queue accounting information is not collected for queues that have the queue attribute ACCTQ set as QMGR. This is the default value.

NONE
The collection of queue accounting information is disabled for all queues, regardless of the queue attribute ACCTQ.

If the queue manager attribute, ACCTQ, is set to NONE, the collection of queue accounting information is disabled for all queues, regardless of the queue attribute ACCTQ.

For example, to enable accounting information collection for the queue, `Q1`, use the following MQSC command:

```
ALTER QLOCAL(Q1) ACCTQ(ON)
```

To enable accounting information collection for all queues that specify the queue attribute ACCTQ as QMGR, use the following MQSC command:

```
ALTER QMGR ACCTQ(ON)
```

MQCONNX options:

Use the **`ConnectOpts`** parameter on the MQCONNX call to modify the collection of both MQI and queue accounting information at the connection level by overriding the effective values of the queue manager attributes ACCTMQI and ACCTQ

The **`ConnectOpts`** parameter can have the following values:

MQCNO_ACCOUNTING_MQI_ENABLED
If the value of the queue manager attribute ACCTMQI is specified as OFF, MQI accounting is enabled for this connection. This is equivalent of the queue manager attribute ACCTMQI being specified as ON.

If the value of the queue manager attribute ACCTMQI is not specified as OFF, this attribute has no effect.

MQCNO_ACCOUNTING_MQI_DISABLED
If the value of the queue manager attribute ACCTMQI is specified as ON, MQI accounting is disabled for this connection. This is equivalent of the queue manager attribute ACCTMQI being specified as OFF.

If the value of the queue manager attribute ACCTMQI is not specified as ON, this attribute has no effect.

MQCNO_ACCOUNTING_Q_ENABLED
If the value of the queue manager attribute ACCTQ is specified as OFF, queue accounting is enabled for this connection. All queues with ACCTQ specified as QMGR, are enabled for queue accounting. This is equivalent of the queue manager attribute ACCTQ being specified as ON.

If the value of the queue manager attribute ACCTQ is not specified as OFF, this attribute has no effect.

MQCNO_ACCOUNTING_Q_DISABLED
If the value of the queue manager attribute ACCTQ is specified as ON, queue accounting is disabled for this connection. This is equivalent of the queue manager attribute ACCTQ being specified as OFF.

If the value of the queue manager attribute ACCTQ is not specified as ON, this attribute has no effect.

These overrides are by disabled by default. To enable them, set the queue manager attribute ACCTCONO to ENABLED. To enable accounting overrides for individual connections use the following MQSC command:

```
ALTER QMGR ACCTCONO(ENABLED)
```

Accounting message generation:

Accounting messages are generated when an application disconnects from the queue manager. Intermediate accounting messages are also written for long running IBM MQ applications.

Accounting messages are generated in either of the following ways when an application disconnects:
- The application issues an MQDISC call
- The queue manager recognises that the application has terminated

Intermediate accounting messages are written for long running IBM MQ applications when the interval since the connection was established or since the last intermediate accounting message that was written exceeds the configured interval. The queue manager attribute, ACCTINT, specifies the time, in seconds, after which intermediate accounting messages can be automatically written. Accounting messages are generated only when the application interacts with the queue manager, so applications that remain connected to the queue manager for long periods without executing MQI requests do not generate accounting messages until the execution of the first MQI request following the completion of the accounting interval.

The default accounting interval is 1800 seconds (30 minutes). For example, to change the accounting interval to 900 seconds (15 minutes) use the following MQSC command:

```
ALTER QMGR ACCTINT(900)
```

Statistics messages

Statistics messages record information about the activities occurring in an IBM MQ system. An statistics messages is a PCF message that contains a number of PCF structures.

Statistics messages are delivered to the system queue (SYSTEM.ADMIN.STATISTICS.QUEUE) at configured intervals, whenever there is some activity.

Statistics messages are in the following categories:

MQI statistics messages
: MQI statistics messages contain information relating to the number of MQI calls made during a configured interval. For example, the information can include the number of MQI calls issued by a queue manager.

Queue statistics messages
: Queue statistics messages contain information relating to the activity of a queue during a configured interval. The information includes the number of messages put on, and retrieved from, the queue, and the total number of bytes processed by a queue.

  Each queue statistics message can contain up to 100 records, with each record relating to the activity per queue for which statistics were collected.

  Statistics messages are recorded only for local queues. If an application makes an MQI call against an alias queue, the statistics data is recorded against the base queue, and, for a remote queue, the statistics data is recorded against the transmission queue.

Channel statistics messages
: Channel statistics messages contain information relating to the activity of a channel during a configured interval. For example the information might be the number of messages transferred by the channel, or the number of bytes transferred by the channel.

  Each channel statistics message contains up to 100 records, with each record relating to the activity per channel for which statistics were collected.

Related reference:
"MQI statistics information" on page 1068
Use the queue manager attribute STATMQI to control the collection of MQI statistics information
"Queue statistics information" on page 1068
Use the queue attribute STATQ and the queue manager attribute STATQ to control the collection of queue statistics information
"Channel statistics information" on page 1069
Use the channel attribute STATCHL to control the collection of channel statistics information. You can also set queue manager attributes to control information collection.

Statistics messages format:

Statistics messages comprise a set of PCF fields that consist of a message descriptor and message data.

Message descriptor

- A statistics message MQMD (message descriptor)

Accounting message data

- A statistics message MQCFH (PCF header)
- Statistics message data that is always returned
- Statistics message data that is returned if available

The statistics message MQCFH (PCF header) contains information about the interval for which the statistics data was recorded.

Statistics message data comprises PCF parameters that store the statistics information. The content of statistics messages depends on the message category as follows:

MQI statistics message
MQI statistics message data consists of a number of PCF parameters, but no PCF groups.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *QStatisticsData* PCF groups.

There is one *QStatisticsData* PCF group for every queue was active in the interval. If more than 100 queues were active in the interval, multiple statistics messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as `MQCFC_LAST`.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *ChlStatisticsData* PCF groups.

There is one *ChlStatisticsData* PCF group for every channel that was active in the interval. If more than 100 channels were active in the interval, multiple statistics messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as `MQCFC_LAST`.

Statistics information collection:

Use queue, queue manager, and channel attributes to control the collection of statistics information

MQI statistics information:

Use the queue manager attribute STATMQI to control the collection of MQI statistics information

To change the value of this attribute, use the MQSC command, `ALTER QMGR` and specify the parameter `STATMQI`. Statistics messages are generated only for queues that are opened after statistics collection has been enabled. The `STATMQI` parameter can have the following values:

ON MQI statistics information is collected for every connection to the queue manager.

OFF MQI statistics information is not collected. This is the default value.

For example, to enable MQI statistics information collection use the following MQSC command:

```
ALTER QMGR STATMQI(ON)
```

Queue statistics information:

Use the queue attribute STATQ and the queue manager attribute STATQ to control the collection of queue statistics information

You can enable or disable queue statistics information collection for individual queues or for multiple queues. To control individual queues, set the queue attribute STATQ. You enable or disable queue statistics information collection at the queue manager level by using the queue manager attribute STATQ. For all queues that have the queue attribute STATQ specified with the value QMGR, queue statistics information collection is controlled at the queue manager level.

Queue statistics are incremented only for operations using IBM MQ MQI Object Handles that were opened after statistics collection has been enabled.

Queue Statistics messages are generated only for queues for which statistics data has been collected in the previous time period.

The same queue can have several put operations and get operations through several Object Handles. Some Object Handles might have been opened before statistics collection was enabled, but others were opened afterwards. Therefore, it is possible for the queue statistics to record the activity of some put operations and get operations, and not all.

To ensure that the Queue Statistics are recording the activity of all applications, you must close and reopen new Object Handles on the queue, or queues, that you are monitoring. The best way to achieve this, is to end and restart all applications after enabling statistics collection.

To change the value of the queue attribute STATQ, use the MQSC command, `ALTER QLOCAL` and specify the parameter `STATQ`. The queue attribute STATQ can have the following values:

ON Queue statistics information is collected for every connection to the queue manager that opens the queue.

OFF Queue statistics information for this queue is not collected.

QMGR
The collection of queue statistics information for this queue is controlled according to the value of the queue manager attribute, STATQ. This is the default value.

To change the value of the queue manager attribute STATQ, use the MQSC command, `ALTER QMGR` and specify the parameter `STATQ`. The queue manager attribute STATQ can have the following values:

ON
Queue statistics information is collected for queues that have the queue attribute STATQ set as QMGR

OFF
Queue statistics information is not collected for queues that have the queue attribute STATQ set as QMGR. This is the default value.

NONE
The collection of queue statistics information is disabled for all queues, regardless of the queue attribute STATQ.

If the queue manager attribute STATQ is set to NONE, the collection of queue statistics information is disabled for all queues, regardless of the queue attribute STATQ.

For example, to enable statistics information collection for the queue, `Q1`, use the following MQSC command:

```
ALTER QLOCAL(Q1) STATQ(ON)
```

To enable statistics information collection for all queues that specify the queue attribute STATQ as QMGR, use the following MQSC command:

```
ALTER QMGR STATQ(ON)
```

Channel statistics information: distributed IBM i

Use the channel attribute STATCHL to control the collection of channel statistics information. You can also set queue manager attributes to control information collection.

You can enable or disable channel statistics information collection for individual channels, or for multiple channels. To control individual channels, you must set the channel attribute STATCHL to enable or disable channel statistic information collection. To control many channels together, you enable or disable channel accounting information collection at the queue manager level by using the queue manager attribute STATCHL. For all channels that have the channel attribute STATCHL specified with the value QMGR, channel accounting information collection is controlled at the queue manager level.

Automatically defined cluster-sender channels are not IBM MQ objects, so do not have attributes in the same way as channel objects. To control automatically defined cluster-sender channels, use the queue manager attribute STATACLS. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel statistics information collection.

You can set channel statistics information collection to one of the three monitoring levels: low, medium or high. You can set the monitoring level at either object level or at the queue manager level. The choice of which level to use is dependent on your system. Collecting statistics information data might require some instructions that are relatively expensive computationally, so to reduce the impact of channel statistics information collection, the medium and low monitoring options measure a sample of the data at regular intervals rather than collecting data all the time. Table 122 on page 1070 summarizes the levels available with channel statistics information collection:

Table 122. Detail level of channel statistics information collection

Level	Description	Usage
Low	Measure a small sample of the data, at regular intervals.	For objects that process a high volume of messages.
Medium	Measure a sample of the data, at regular intervals.	For most objects.
High	Measure all data, at regular intervals.	For objects that process only a few messages per second, on which the most current information is important.

To change the value of the channel attribute STATCHL, use the MQSC command, `ALTER CHANNEL` and specify the parameter `STATCHL`.

To change the value of the queue manager attribute STATCHL, use the MQSC command, `ALTER QMGR` and specify the parameter `STATCHL`.

To change the value of the queue manager attribute STATACLS, use the MQSC command, `ALTER QMGR` and specify the parameter `STATACLS`.

The channel attribute, STATCHL, can have the following values:

LOW Channel statistics information is collected with a low level of detail.

MEDIUM
Channel statistics information is collected with a medium level of detail.

HIGH Channel statistics information is collected with a high level of detail.

OFF Channel statistics information is not collected for this channel.

QMGR
The channel attribute is set as QMGR. The collection of statistics information for this channel is controlled by the value of the queue manager attribute, STATCHL.

This is the default value.

On z/OS, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

The queue manager attribute, STATCHL, can have the following values:

LOW Channel statistics information is collected with a low level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

MEDIUM
Channel statistics information is collected with a medium level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

HIGH Channel statistics information is collected with a high level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

OFF Channel statistics information is not collected for all channels that have the channel attribute STATCHL set as QMGR.

This is the default value.

NONE
The collection of channel statistics information is disabled for all channel, regardless of the channel attribute STATCHL.

On z/OS, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

The queue manager attribute, STATACLS, can have the following values:

LOW Statistics information is collected with a low level of detail for automatically defined cluster-sender channels.

MEDIUM
Statistics information is collected with a medium level of detail for automatically defined cluster-sender channels.

HIGH Statistics information is collected with a high level of detail for automatically defined cluster-sender channels.

OFF Statistics information is not collected for automatically defined cluster-sender channels.

QMGR
The collection of statistics information for automatically defined cluster-sender channels is controlled by the value of the queue manager attribute, STATCHL.

This is the default value.

On z/OS, this parameter simply turns on statistics data collection, regardless of the value you select. Specifying LOW, MEDIUM, or HIGH makes no difference to your results.

For example, to enable statistics information collection, with a medium level of detail, for the sender channel `QM1.TO.QM2`, use the following MQSC command:

```
ALTER CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) STATCHL(MEDIUM)
```

To enable statistics information collection, at a medium level of detail, for all channels that specify the channel attribute STATCHL as QMGR, use the following MQSC command:

```
ALTER QMGR STATCHL(MEDIUM)
```

To enable statistics information collection, at a medium level of detail, for all automatically defined cluster-sender channels, use the following MQSC command:

```
ALTER QMGR STATACLS(MEDIUM)
```

Statistics message generation:

Statistics messages are generated at configured intervals, and when a queue manager shuts down in a controlled fashion.

The configured interval is controlled by the STATINT queue manager attribute, which specifies the interval, in seconds, between the generation of statistics messages. The default statistics interval is 1800 seconds (30 minutes). To change the statistics interval, use the MQSC command `ALTER QMGR` and specify the `STATINT` parameter. For example, to change the statistics interval to 900 seconds (15 minutes) use the following MQSC command:

```
ALTER QMGR STATINT(900)
```

To write the currently collected statistics data to the statistics queue before the statistics collection interval is due to expire, use the MQSC command `RESET QMGR TYPE(STATISTICS)`. Issuing this command causes the collected statistics data to be written to the statistics queue and a new statistics data collection interval to begin.

Displaying accounting and statistics information

To use the information recorded in accounting and statistics messages, run an application such as the **amqsmon** sample program to transform the recorded information into a suitable format

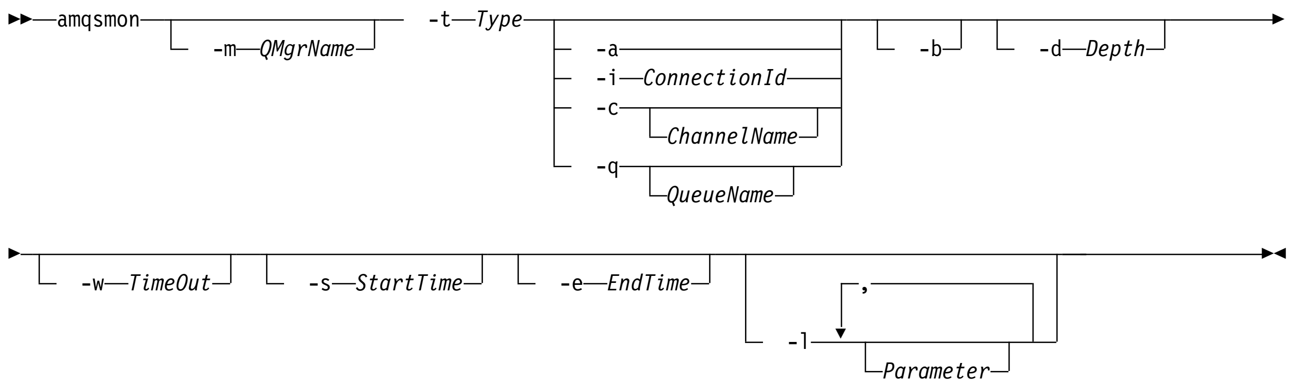
Accounting and statistics messages are written to the system accounting and statistics queues. **amqsmon** is a sample program supplied with IBM MQ that processes messages from the accounting and statistics queues and displays the information to the screen in a readable form.

Because **amqsmon** is a sample program, you can use the supplied source code as template for writing your own application to process accounting or statistics messages, or modify the **amqsmon** source code to meet your own particular requirements.

amqsmon (Display formatted monitoring information):

Use the **amqsmon** sample program to display in a readable format the information contained within accounting and statistics messages. The **amqsmon** program reads accounting messages from the accounting queue, SYSTEM.ADMIN.ACCOUNTING.QUEUE. and reads statistics messages from the statistics queue, SYSTEM.ADMIN.STATISTICS.QUEUE.

Syntax



Required parameters

`-t` *Type*
: The type of messages to process. Specify *Type* as one of the following:

accounting
: Accounting records are processed. Messages are read from the system queue, SYSTEM.ADMIN.ACCOUNTING.QUEUE.

statistics
: Statistics records are processed. Messages are read from the system queue, SYSTEM.ADMIN.STATISTICS.QUEUE.

Optional Parameters

-m *QMgrName*
: The name of the queue manager from which accounting or statistics messages are to be processed.

If you do not specify this parameter, the default queue manager is used.

-a Process messages containing MQI records only.

Only display MQI records. Messages not containing MQI records will always be left on the queue they were read from.

-q *`QueueName`*
QueueName is an optional parameter.

If *QueueName* is not supplied:	Displays queue accounting and queue statistics records only.
If *QueueName* is supplied:	Displays queue accounting and queue statistics records for the queue specified by *QueueName* only. If *`-b`* is not specified then the accounting and statistics messages from which the records came are discarded. Since accounting and statistics messages can also contain records from other queues, if *`-b`* is not specified then unseen records can be discarded.

-c *`ChannelName`*
ChannelName is an optional parameter.

If *ChannelName* is not supplied:	Displays channel statistics records only.
If *ChannelName* is supplied:	Displays channel statistics records for the channel specified by *ChannelName* only. If *`-b`* is not specified then the statistics messages from which the records came are discarded. Since statistics messages can also contain records from other channels, if *`-b`* is not specified then unseen records can be discarded.

This parameter is available when displaying statistics messages only, (*-t statistics*).

-i *`ConnectionId`*
Displays records related to the connection identifier specified by *ConnectionId* only.

This parameter is available when displaying accounting messages only, (*-t accounting*).

If *`-b`* is not specified then the statistics messages from which the records came are discarded. Since statistics messages can also contain records from other channels, if *`-b`* is not specified then unseen records can be discarded.

-b Browse messages.

Messages are retrieved non-destructively.

-d *`Depth`*
The maximum number of messages that can be processed.

If you do not specify this parameter, then an unlimited number of messages can be processed.

-w *`TimeOut`*
Time maximum number of seconds to wait for a message to become available.

If you do not specify this parameter, **amqsmon** will end once there are no more messages to process.

-s *`StartTime`*
Process messages put after the specified *StartTime* only.

StartTime is specified in the format `yyyy-mm-dd hh.mm.ss`. If a date is specified without a time, then the time will default to `00.00.00` on the date specified. Times are in GMT.

For the effect of not specifying this parameter, see Note 1.

-e *`EndTime`*
Process messages put before the specified *EndTime* only.

The *EndTime* is specified in the format `yyyy-mm-dd hh.mm.ss`. If a date is specified without a time, then the time will default to `23.59.59` on the date specified. Times are in GMT.

For the effect of not specifying this parameter, see Note 1.

-l *Parameter*

Only display the selected fields from the records processed. *Parameter* is a comma-separated list of integer values, with each integer value mapping to the numeric constant of a field, see amqsmon example 5.

If you do not specify this parameter, then all available fields are displayed.

Note:

1. If you do not specify *-s StartTime* or *-e EndTime*, the messages that can be processed are not restricted by put time.

amqsmon examples:

Use this page to view examples of running the amqsmon (Display formatted monitoring information) sample program

1. The following command displays all MQI statistics messages from queue manager `saturn.queue.manager`:

   ```
   amqsmon -m saturn.queue.manager -t statistics -a
   ```

   The output from this command follows:

   ```
   RecordType: MQIStatistics
   QueueManager: 'saturn.queue.manager'
   IntervalStartDate: '2005-04-30'
   IntervalStartTime: '15.09.02'
   IntervalEndDate: '2005-04-30'
   IntervalEndTime: '15.39.02'
   CommandLevel: 600
   ConnCount: 23
   ConnFailCount: 0
   ConnHighwater: 8
   DiscCount: [17, 0, 0]
   OpenCount: [0, 80, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0]
   OpenFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   CloseCount: [0, 73, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
   CloseFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   InqCount: [4, 2102, 0, 0, 0, 46, 0, 0, 0, 0, 0, 0, 0]
   InqFailCount: [0, 31, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   SetCount: [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
   SetFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   PutCount: [26, 1]
   PutFailCount: 0
   Put1Count: [40, 0]
   Put1FailCount: 0
   PutBytes: [57064, 12320]
   GetCount: [18, 1]
   GetBytes: [52, 12320]
   GetFailCount: 2254
   BrowseCount: [18, 60]
   BrowseBytes: [23784, 30760]
   BrowseFailCount: 9
   CommitCount: 0
   CommitFailCount: 0
   BackCount: 0
   ExpiredMsgCount: 0
   PurgeCount: 0
   ```

2. The following command displays all queue statistics messages for queue `LOCALQ` on queue manager `saturn.queue.manager`:

   ```
   amqsmon -m saturn.queue.manager -t statistics -q LOCALQ
   ```

   The output from this command follows:

```
RecordType: QueueStatistics
QueueManager: 'saturn.queue.manager'
IntervalStartDate: '2005-04-30'
IntervalStartTime: '15.09.02'
IntervalEndDate: '2005-04-30'
IntervalEndTime: '15.39.02'
CommandLevel: 600
ObjectCount: 3
QueueStatistics:
  QueueName: 'LOCALQ'
  CreateDate: '2005-03-08'
  CreateTime: '17.07.02'
  QueueType: Predefined
  QueueDefinitionType: Local
  QMinDepth: 0
  QMaxDepth: 18
  AverageQueueTime: [29827281, 0]
  PutCount: [26, 0]
  PutFailCount: 0
  Put1Count: [0, 0]
  Put1FailCount: 0
  PutBytes: [88, 0]
  GetCount: [18, 0]
  GetBytes: [52, 0]
  GetFailCount: 0
  BrowseCount: [0, 0]
  BrowseBytes: [0, 0]
  BrowseFailCount: 1
  NonQueuedMsgCount: 0
  ExpiredMsgCount: 0
  PurgedMsgCount: 0
```

3. The following command displays all of the statistics messages recorded since 15:30 on 30 April 2005 from queue manager `saturn.queue.manager`.

   ```
   amqsmon -m saturn.queue.manager -t statistics -s "2005-04-30 15.30.00"
   ```

   The output from this command follows:

   ```
   RecordType: MQIStatistics
   QueueManager: 'saturn.queue.manager'
   IntervalStartDate: '2005-04-30'
   IntervalStartTime: '15.09.02'
   IntervalEndDate: '2005-04-30'
   IntervalEndTime: '15.39.02'
   CommandLevel: 600
   ConnCount: 23
   ConnFailCount: 0
   ConnHighwater: 8
   DiscCount: [17, 0, 0]
   OpenCount: [0, 80, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0]
     ...
   RecordType: QueueStatistics
   QueueManager: 'saturn.queue.manager'
   IntervalStartDate: '2005-04-30'
   IntervalStartTime: '15.09.02'
   IntervalEndDate: '2005-04-30'
   IntervalEndTime: '15.39.02'
   CommandLevel: 600
   ObjectCount: 3
   QueueStatistics: 0
     QueueName: 'LOCALQ'
     CreateDate: '2005-03-08'
     CreateTime: '17.07.02'
     QueueType: Predefined
       ...
   QueueStatistics: 1
     QueueName: 'SAMPLEQ'
   ```

```
     CreateDate: '2005-03-08'
     CreateTime: '17.07.02'
     QueueType: Predefined
       ...
```

4. The following command displays all accounting messages recorded on 30 April 2005 from queue manager `saturn.queue.manager`:

   ```
   amqsmon -m saturn.queue.manager -t accounting -s "2005-04-30" -e "2005-04-30"
   ```

   The output from this command follows:

   ```
      RecordType: MQIAccounting
      QueueManager: 'saturn.queue.manager'
      IntervalStartDate: '2005-04-30'
      IntervalStartTime: '15.09.29'
      IntervalEndDate: '2005-04-30'
      IntervalEndTime: '15.09.30'
      CommandLevel: 600
      ConnectionId: x'414d5143545245563120202020202020208d0b3742010a0020'
      SeqNumber: 0
      ApplicationName: 'amqsput'
      ApplicationPid: 8572
      ApplicationTid: 1
      UserId: 'admin'
      ConnDate: '2005-03-16'
      ConnTime: '15.09.29'
      DiscDate: '2005-03-16'
      DiscTime: '15.09.30'
      DiscType: Normal
      OpenCount: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      OpenFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      CloseCount: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      CloseFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      PutCount: [1, 0]
      PutFailCount: 0
      PutBytes: [4, 0]
      GetCount: [0, 0]
      GetFailCount: 0
      GetBytes: [0, 0]
      BrowseCount: [0, 0]
      BrowseFailCount: 0
      BrowseBytes: [0, 0]
      CommitCount: 0
      CommitFailCount: 0
      BackCount: 0
      InqCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      InqFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      SetCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      SetFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

      RecordType: MQIAccounting
      QueueManager: 'saturn.queue.manager'
      IntervalStartDate: '2005-03-16'
      IntervalStartTime: '15.16.22'
      IntervalEndDate: '2005-03-16'
      IntervalEndTime: '15.16.22'
      CommandLevel: 600
      ConnectionId: x'414d5143545245563120202020202020208d0b3742010c0020'
      SeqNumber: 0
      ApplicationName: 'runmqsc'
      ApplicationPid: 8615
      ApplicationTid: 1
          ...
   ```

5. The following command browses the accounting queue and displays the application name and connection identifier of every application for which MQI accounting information is available:

   ```
   amqsmon -m saturn.queue.manager -t accounting -b -a -l 7006,3024
   ```

The output from this command follows:

```
ConnectionId: x'414d5143545245563120202020202020208d0b374203090020'
ApplicationName: 'runmqsc'

ConnectionId: x'414d5143545245563120202020202020208d0b3742010a0020'
ApplicationName: 'amqsput'

ConnectionId: x'414d5143545245563120202020202020208d0b3742010c0020'
ApplicationName: 'runmqsc'

ConnectionId: x'414d5143545245563120202020202020208d0b3742010d0020'
ApplicationName: 'amqsput'

ConnectionId: x'414d5143545245563120202020202020208d0b3742150d0020'
ApplicationName: 'amqsget'

5 Records Processed.
```

Accounting and statistics message reference

Use this page to obtain an overview of the format of accounting and statistics messages and the information returned in these messages

Accounting and statistics message messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the MQI operations performed by IBM MQ applications, or information about the activities occurring in an IBM MQ system.

Message descriptor

- An MQMD structure

Message data

- A PCF header (MQCFH)
- Accounting or statistics message data that is always returned
- Accounting or statistics message data that is returned if available

Accounting and statistics message format:

Use this page as an example of the structure of an MQI accounting message

Table 123. MQI accounting message structure

MQMD structure	Accounting message header MQCFH structure	MQI accounting message data [1]
Structure identifier	Structure type	Queue manager
Structure version	Structure length	Interval start date
Report options	Structure version	Interval start time
Message type	Command identifier	Interval end date
Expiration time	Message sequence number	Interval end time
Feedback code	Control options	Command level
Encoding	Completion code	Connection identifier
Coded character set ID	Reason code	Sequence number
Message format	Parameter count	Application name
Message priority		Application process identifier
Persistence		Application thread identifier
Message identifier		User identifier
Correlation identifier		Connection date
Backout count		Connection time
Reply-to queue		Connection name
Reply-to queue manager		Channel name
User identifier		Disconnect date
Accounting token		Disconnect time
Application identity data		Disconnect type
Application type		Open count
Application name		Open fail count
Put date		Close count
Put time		Close fail count
Application origin data		Put count
Group identifier		Put fail count
Message sequence number		Put1 count
Offset		Put1 fail count
Message flags		Put bytes
Original length		Get count
		Get fail count
		Get bytes
		Browse count
		Browse fail count
		Browse bytes
		Commit count
		Commit fail count
		Backout count
		Inquire count
		Inquire fail count
		Set count
		Set fail count

Note:

1. The parameters shown are those returned for an MQI accounting message. The actual accounting or statistics message data depends on the message category.

Accounting and statistics message MQMD (message descriptor):

Use this page to understand the differences between the message descriptor of accounting and statistics messages and the message descriptor of event messages

The parameters and values in the message descriptor of accounting and statistics message are the same as in the message descriptor of event messages, with the following exception:

Format

Description:	Format name of message data.
Data type:	MQCHAR8.
Value:	**MQFMT_ADMIN** Admin message.

Some of the parameters contained in the message descriptor of accounting and statistics message contain fixed data supplied by the queue manager that generated the message.

The MQMD also specifies the name of the queue manager (truncated to 28 characters) that put the message, and the date and time when the message was put on the accounting, or statistics, queue.

Message data in accounting and statistics messages:

The message data in accounting and statistics messages is based on the programmable command format (PCF), which is used in PCF command inquiries and responses. The message data in accounting and statistics messages consists of a PCF header (MQCFH) and an accounting or statistics report.

Accounting and statistics message MQCFH (PCF header)

The message header of accounting and statistics messages is an MQCFH structure. The parameters and values in the message header of accounting and statistics message are the same as in the message header of event messages, with the following exceptions:

Command

Description:	Command identifier. This identifies the accounting or statistics message category.
Data type:	MQLONG.
Values:	**MQCMD_ACCOUNTING_MQI** MQI accounting message. **MQCMD_ACCOUNTING_Q** Queue accounting message. **MQCMD_STATISTICS_MQI** MQI statistics message. **MQCMD_STATISTICS_Q** Queue statistics message. **MQCMD_STATISTICS_CHANNEL** Channel statistics message.

Version

Description:	Structure version number.
Data type:	MQLONG.
Value:	**MQCFH_VERSION_3** Version-3 for accounting and statistics messages.

Accounting and statistics message data

The content of accounting and statistics message data is dependent on the category of the accounting or statistics message, as follows:

MQI accounting message
MQI accounting message data consists of a number of PCF parameters, but no PCF groups.

Queue accounting message
Queue accounting message data consists of a number of PCF parameters, and in the range 1 through 100 *QAccountingData* PCF groups.

MQI statistics message
MQI statistics message data consists of a number of PCF parameters, but no PCF groups.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *QStatisticsData* PCF groups.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and in the range 1 through 100 *ChlStatisticsData* PCF groups.

MQI accounting message data:

Use this page to view the structure of an MQI accounting message

Message name:	MQI accounting message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.ACCOUNTING.QUEUE.

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time of the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL
Data type:	MQCFIN
Returned:	Always

ConnectionId

Description:	The connection identifier for the IBM MQ connection
Identifier:	MQBACF_CONNECTION_ID
Data type:	MQCFBS
Maximum length:	MQ_CONNECTION_ID_LENGTH
Returned:	Always

SeqNumber

Description:	The sequence number. This value is incremented for each subsequent record for long running connections.
Identifier:	MQIACF_SEQUENCE_NUMBER
Data type:	MQCFIN
Returned:	Always

ApplicationName

Description:	The name of the application. The contents of this field are equivalent to the contents of the *PutApplName* field in the message descriptor.
Identifier:	MQCACF_APPL_NAME
Data type:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH
Returned:	Always

ApplicationPid

Description:	The operating system process identifier of the application
Identifier:	MQIACF_PROCESS_ID
Data type:	MQCFIN
Returned:	Always

ApplicationTid

Description:	The IBM MQ thread identifier of the connection in the application
Identifier:	MQIACF_THREAD_ID
Data type:	MQCFIN
Returned:	Always

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Data type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH
Returned:	Always

ConnDate

Description:	Date of MQCONN operation
Identifier:	MQCAMO_CONN_DATE
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

ConnTime

Description:	Time of MQCONN operation
Identifier:	MQCAMO_CONN_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

ConnName

Description:	Connection name for client connection
Identifier:	MQCACH_CONNECTION_NAME
Data type:	MQCFST
Maximum length:	MQ_CONN_NAME_LENGTH
Returned:	When available

ChannelName

Description:	Channel name for client connection
Identifier:	MQCACH_CHANNEL_NAME
Data type:	MQCFST
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	When available

DiscDate

Description:	Date of MQDISC operation
Identifier:	MQCAMO_DISC_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	When available

DiscTime

Description:	Time of MQDISC operation
Identifier:	MQCAMO_DISC_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

DiscType

Description:	Type of disconnect
Identifier:	MQIAMO_DISC_TYPE
Data type:	MQCFIN
Values:	The possible values are: **MQDISCONNECT_NORMAL** Requested by application; **MQDISCONNECT_IMPLICIT** Abnormal application termination; **MQDISCONNECT_Q_MGR** Connection broken by queue manager
Returned:	When available

OpenCount

Description:	The number of objects opened. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_OPENS
Data type:	MQCFIL
Returned:	When available

OpenFailCount

Description:	The number of unsuccessful attempts to open an object. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_OPENS_FAILED
Data type:	MQCFIL
Returned:	When available

CloseCount

Description:	The number of objects closed. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES
Data type:	MQCFIL
Returned:	When available

CloseFailCount

Description:	The number of unsuccessful attempts to close an object. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES_FAILED
Data type:	MQCFIL
Returned:	When available

PutCount

Description:	The number persistent and nonpersistent messages successfully put to a queue, with the exception of messages put using the MQPUT1 call. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUTS
Data type:	MQCFIL
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message
Identifier:	MQIAMO_PUTS_FAILED
Data type:	MQCFIN
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUT1S
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

PutBytes

Description:	The number bytes written using put calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUT_BYTES
Data type:	MQCFIL64
Returned:	When available

GetCount

Description:	The number of successful destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_GETS
Data type:	MQCFIL
Returned:	When available

GetFailCount

Description:	The number of failed destructive MQGET calls
Identifier:	MQIAMO_GETS_FAILED
Data type:	MQCFIN
Returned:	When available

GetBytes

Description:	Total number of bytes retrieved for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_GET_BYTES
Data type:	MQCFIL64
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSES
Data type:	MQCFIL
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive MQGET calls
Identifier:	MQIAMO_BROWSES_FAILED
Data type:	MQCFIN
Returned:	When available

BrowseBytes

Description:	Total number of bytes browsed for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_BROWSE_BYTES
Data type:	MQCFIL64
Returned:	When available

CommitCount

Description:	The number of successful transactions. This number includes those transactions committed implicitly by the connected application. Commit requests where there is no outstanding work are included in this count.
Identifier:	MQIAMO_COMMITS
Data type:	MQCFIN
Returned:	When available

CommitFailCount

Description:	The number of unsuccessful attempts to complete a transaction
Identifier:	MQIAMO_COMMITS_FAILED
Data type:	MQCFIN
Returned:	When available

BackCount

Description:	The number of backouts processed, including implicit backouts due to abnormal disconnection
Identifier:	MQIAMO_BACKOUTS
Data type:	MQCFIN
Returned:	When available

InqCount

Description:	The number of successful objects inquired upon. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS
Data type:	MQCFIL
Returned:	When available

InqFailCount

Description:	The number of unsuccessful object inquire attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS_FAILED
Data type:	MQCFIL
Returned:	When available

SetCount

Description:	The number of successful MQSET calls. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS
Data type:	MQCFIL
Returned:	When available

SetFailCount

Description:	The number of unsuccessful MQSET calls. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS_FAILED
Data type:	MQCFIL
Returned:	When available

SubCountDur

Description:	The number of succesful subscribe requests which created, altered or resumed durable subscriptions. This is an array of values indexed by the type of operation 0 = The number of subscriptions created 1 = The number of subscriptions altered 2 = The number of subscriptions resumed
Identifier:	MQIAMO_SUBS_DUR
Data type:	MQCFIL
Returned:	When available.

SubCountNDur

Description:	The number of succesful subscribe requests which created, altered or resumed non-durable subscriptions. This is an array of values indexed by the type of operation 0 = The number of subscriptions created 1 = The number of subscriptions altered 2 = The number of subscriptions resumed
Identifier:	MQIAMO_SUBS_NDUR
Data type:	MQCFIL
Returned:	When available.

SubFailCount

Description:	The number of unsuccessful Subscribe requests.
Identifier:	MQIAMO_SUBS_FAILED
Data type:	MQCFIN
Returned:	When available.

UnsubCountDur

Description:	The number of succesful unsubscribe requests for durable subscriptions. This is an array of values indexed by the type of operation 0 - The subscription was closed but not removed 1 - The subscription was closed and removed
Identifier:	MQIAMO_UNSUBS_DUR
Data type:	MQCFIL
Returned:	When available.

UnsubCountNDur

Description:	The number of succesful unsubscribe requests for durable subscriptions. This is an array of values indexed by the type of operation 0 - The subscription was closed but not removed 1 - The subscription was closed and removed
Identifier:	MQIAMO_UNSUBS_NDUR
Data type:	MQCFIL
Returned:	When available.

UnsubFailCount

Description:	The number of unsuccessful unsubscribe requests.
Identifier:	MQIAMO_UNSUBS_FAILED
Data type:	MQCFIN
Returned:	When available.

SubRqCount

Description:	The number of successful MQSUBRQ requests.
Identifier:	MQIAMO_SUBRQS
Data type:	MQCFIN
Returned:	When available.

SubRqFailCount

Description:	The number of unsuccessful MQSUB requests.
Identifier:	MQIAMO_SUBRQS_FAILED
Data type:	MQCFIN
Returned:	When available.

CBCount

Description:	The number of successful MQCB requests. This is an array of values indexed by the type of operation 0 - A callback was created or altered 1 - A callback was removed 2 - A callback was resumed 3 - A callback was suspended
Identifier:	MQIAMO_CBS
Data type:	MQCFIN
Returned:	When available.

CBFailCount

Description:	The number of unsuccessful MQCB requests.
Identifier:	MQIAMO_CBS_FAILED
Data type:	MQCFIN
Returned:	When available.

CtlCount

Description:	The number of successful MQCTL requests. This is an array of values indexed by the type of operation 0 - The connection was started 1 - The connection was stopped 2 - The connection was resumed 3 - The connection was suspended
Identifier:	MQIAMO_CTLS
Data type:	MQCFIL
Returned:	When available.

CtlFailCount

Description:	The number of unsuccessful MQCTL requests.
Identifier:	MQIAMO_CTLS_FAILED
Data type:	MQCFIN
Returned:	When available.

StatCount

Description:	The number of successful MQSTAT requests.
Identifier:	MQIAMO_STATS.
Data type:	MQCFIN
Returned:	When available.

StatFailCount

Description:	The number of unsuccessful MQSTAT requests.
Identifier:	MQIAMO_STATS_FAILED
Data type:	MQCFIN
Returned:	When available.

PutTopicCount

Description:	The number persistent and nonpersistent messages successfully put to a topic, with the exception of messages put using the MQPUT1 call. This parameter is an integer list indexed by persistence value, see Reference note 2. Note: Messages put using a queue alias which resolve to a topic are included in this value.
Identifier:	MQIAMO_TOPIC_PUTS
Data type:	MQCFIL
Returned:	When available.

PutTopicFailCount

Description:	The number of unsuccessful attempts to put a message to a topic.
Identifier:	MQIAMO_TOPIC_PUTS_FAILED
Data type:	MQCFIN
Returned:	When available.

Put1TopicCount

Description:	The number of persistent and nonpersistent messages successfully put to a topic using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2. Note: Messages put using a queue alias which resolve to a topic are included in this value.
Identifier:	MQIAMO_TOPIC_PUT1S
Data type:	MQCFIL
Returned:	When available.

Put1TopicFailCount

Description:	The number of unsuccessful attempts to put a message to a topic using MQPUT1 calls.
Identifier:	MQIAMO_TOPIC_PUT1S_FAILED
Data type:	MQCFIN
Returned:	When available.

PutTopicBytes

Description:	The number bytes written using put calls for persistent and nonpersistent messages which resolve to a publish operation. This is number of bytes put by the application and not the resultant number of bytes delivered to subscribers. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_TOPIC_PUT_BYTES
Data type:	MQCFIL64
Returned:	When available.

Queue accounting message data:

Use this page to view the structure of a queue accounting message

Message name:	Queue accounting message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.ACCOUNTING.QUEUE.

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time of the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL
Data type:	MQCFIN
Returned:	Always

ConnectionId

Description:	The connection identifier for the IBM MQ connection
Identifier:	MQBACF_CONNECTION_ID
Data type:	MQCFBS
Maximum length:	MQ_CONNECTION_ID_LENGTH
Returned:	Always

SeqNumber

Description:	The sequence number. This value is incremented for each subsequent record for long running connections.
Identifier:	MQIACF_SEQUENCE_NUMBER
Data type:	MQCFIN
Returned:	Always

ApplicationName

Description:	The name of the application. The contents of this field are equivalent to the contents of the PutApplName field in the message descriptor.
Identifier:	MQCACF_APPL_NAME
Data type:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH
Returned:	Always

ApplicationPid

Description:	The operating system process identifier of the application
Identifier:	MQIACF_PROCESS_ID
Data type:	MQCFIN
Returned:	Always

ApplicationTid

Description:	The IBM MQ thread identifier of the connection in the application
Identifier:	MQIACF_THREAD_ID
Data type:	MQCFIN
Returned:	Always

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Data type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH
Returned:	Always

ObjectCount

Description:	The number of queues accessed in the interval for which accounting data has been recorded. This value is set to the number of *QAccountingData* PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Data type:	MQCFIN
Returned:	Always

QAccountingData

Description:	Grouped parameters specifying accounting details for a queue
Identifier:	MQGACF_Q_ACCOUNTING_DATA
Data type:	MQCFGR

Parameters in group:	*QName* *CreateDate* *CreateTime* *QType* *QDefinitionType* *OpenCount* *OpenDate* *OpenTime* *CloseDate* *CloseTime* *PutCount* *PutFailCount* *Put1Count* *Put1FailCount* *PutBytes* *PutMinBytes* *PutMaxBytes* *GetCount* *GetFailCount* *GetBytes* *GetMinBytes* *GetMaxBytes* *BrowseCount* *BrowseFailCount* *BrowseBytes* *BrowseMinBytes* *BrowseMaxBytes* *TimeOnQMin* *TimeOnQAvg* *TimeOnQMax*
Returned:	Always

QName

Description:	The name of the queue
Identifier:	MQCA_Q_NAME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	When available

CreateDate

Description:	The date the queue was created
Identifier:	MQCA_CREATION_DATE
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_DATE_LENGTH
Returned:	When available

CreateTime

Description:	The time the queue was created
Identifier:	MQCA_CREATION_TIME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

QType

Description:	The type of the queue
Identifier:	MQIA_Q_TYPE
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Value:	MQQT_LOCAL
Returned:	When available

QDefinitionType

Description:	The queue definition type
Identifier:	MQIA_DEFINITION_TYPE
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Values:	Possible values are: **MQQDT_PREDEFINED** **MQQDT_PERMANENT_DYNAMIC** **MQQDT_TEMPORARY_DYNAMIC**
Returned:	When available

OpenCount

Description:	The number of times this queue was opened by the application in this interval
Identifier:	MQIAMO_OPENS
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

OpenDate

Description:	The date the queue was first opened in this recording interval. If the queue was already open at the start of this interval, this value reflects the date the queue was originally opened.
Identifier:	MQCAMO_OPEN_DATE
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

OpenTime

Description:	The time the queue was first opened in this recording interval. If the queue was already open at the start of this interval, this value reflects the time the queue was originally opened.
Identifier:	MQCAMO_OPEN_TIME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

CloseDate

Description:	The date of the final close of the queue in this recording interval. If the queue is still open then the value is not returned.
Identifier:	MQCAMO_CLOSE_DATE
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

CloseTime

Description:	The time of final close of the queue in this recording interval. If the queue is still open then the value is not returned.
Identifier:	MQCAMO_CLOSE_TIME
Data type:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to the queue, with the exception of MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUTS
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message, with the exception of MQPUT1 calls
Identifier:	MQIAMO_PUTS_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUT1S
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

PutBytes

Description:	The total number of bytes put for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUT_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

PutMinBytes

Description:	The smallest persistent and nonpersistent message size placed on the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUT_MIN_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

PutMaxBytes

Description:	The largest persistent and nonpersistent message size placed on the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUT_MAX_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

GeneratedMsgCount

Description:	The number of generated messages. Generated messages are • Trigger messages • Queue Depth Hi Events • Queue Depth Low Events
Identifier:	MQIAMO_GENERATED_MSGS
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

GetCount

Description:	The number of successful destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_GETS
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

GetFailCount

Description:	The number of failed destructive MQGET calls
Identifier:	MQIAMO_GETS_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

GetBytes

Description:	The number of bytes read in destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_GET_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

GetMinBytes

Description:	The size of the smallest persistent and nonpersistent message retrieved rom the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_GET_MIN_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

GetMaxBytes

Description:	The size of the largest persistent and nonpersistent message retrieved rom the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_GET_MAX_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive MQGET calls
Identifier:	MQIAMO_BROWSES_FAILED
Data type:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseBytes

Description:	The number of bytes read in non-destructive MQGET calls that returned persistent messages
Identifier:	MQIAMO64_BROWSE_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseMinBytes

Description:	The size of the smallest persistent and nonpersistent message browsed from the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSE_MIN_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseMaxBytes

Description:	The size of the largest persistent and nonpersistent message browsed from the queue. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_BROWSE_MAX_BYTES
Data type:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

CBCount

Description:	The number of successful MQCB requests. This is an array of values indexed by the type of operation 0 - A callback was created or altered 1 - A callback was removed 2 - A callback was resumed 3 - A callback was suspended
Identifier:	MQIAMO_CBS
Data type:	MQCFIN
Returned:	When available.

CBFailCount

Description:	The number of unsuccessful MQCB requests.
Identifier:	MQIAMO_CBS_FAILED
Data type:	MQCFIN
Returned:	When available.

TimeOnQMin

Description:	The shortest time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_Q_TIME_MIN
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQAvg

Description:	The average time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_Q_TIME_AVG
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQMax

Description:	The longest time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_Q_TIME_MAX
Data type:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

MQI statistics message data:

Use this page to view the structure of an MQI statistics message

Message name:	MQI statistics message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

QueueManager

Description:	Name of the queue manager.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

IntervalStartDate

Description:	The date at the start of the monitoring period.
Identifier:	MQCAMO_START_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH
Returned:	Always.

IntervalStartTime

Description:	The time at the start of the monitoring period.
Identifier:	MQCAMO_START_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

IntervalEndDate

Description:	The date at the end of the monitoring period.
Identifier:	MQCAMO_END_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH
Returned:	Always.

IntervalEndTime

Description:	The time at the end of the monitoring period.
Identifier:	MQCAMO_END_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

CommandLevel

Description:	The queue manager command level.
Identifier:	MQIA_COMMAND_LEVEL.
Data type:	MQCFIN.
Returned:	Always.

ConnCount

Description:	The number of successful connections to the queue manager.
Identifier:	MQIAMO_CONNS.
Data type:	MQCFIN.
Returned:	When available.

ConnFailCount

Description:	The number of unsuccessful connection attempts.
Identifier:	MQIAMO_CONNS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

ConnsMax

Description:	The maximum number of concurrent connections in the recording interval.
Identifier:	MQIAMO_CONNS_MAX.
Data type:	MQCFIN.
Returned:	When available.

DiscCount

Description:	The number of disconnects from the queue manager. This is an integer array, indexed by the following constants: • MQDISCONNECT_NORMAL • MQDISCONNECT_IMPLICIT • MQDISCONNECT_Q_MGR
Identifier:	MQIAMO_DISCS.
Data type:	MQCFIL.
Returned:	When available.

OpenCount

Description:	The number of objects successfully opened. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_OPENS.
Data type:	MQCFIL.
Returned:	When available.

OpenFailCount

Description:	The number of unsuccessful open object attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_OPENS_FAILED.
Data type:	MQCFIL.
Returned:	When available.

CloseCount

Description:	The number of objects successfully closed. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES.
Data type:	MQCFIL.
Returned:	When available.

CloseFailCount

Description:	The number of unsuccessful close object attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_CLOSES_FAILED.
Data type:	MQCFIL.
Returned:	When available.

InqCount

Description:	The number of objects successfully inquired upon. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS.
Data type:	MQCFIL.
Returned:	When available.

InqFailCount

Description:	The number of unsuccessful object inquire attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_INQS_FAILED.
Data type:	MQCFIL.
Returned:	When available.

SetCount

Description:	The number of objects successfully updated (SET). This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS.
Data type:	MQCFIL.
Returned:	When available.

SetFailCount

Description:	The number of unsuccessful SET attempts. This parameter is an integer list indexed by object type, see Reference note 1.
Identifier:	MQIAMO_SETS_FAILED.
Data type:	MQCFIL.
Returned:	When available.

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to a queue, with the exception of MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO_PUTS.
Data type:	MQCFIL.
Returned:	When available.

PutFailCount

Description:	The number of unsuccessful put message attempts.
Identifier:	MQIAMO_PUTS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to a queue using MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Reference note 2
Identifier:	MQIAMO_PUT1S.
Data type:	MQCFIL.
Returned:	When available.

Put1FailCount

Description:	The number of unsuccessful attempts to put a persistent and nonpersistent message to a queue using MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Reference note 2
Identifier:	MQIAMO_PUT1S_FAILED.
Data type:	MQCFIL.
Returned:	When available.

PutBytes

Description:	The number bytes for persistent and nonpersistent messages written in using put requests. This parameter is an integer list indexed by persistence value, see Reference note 2
Identifier:	MQIAMO64_PUT_BYTES.
Data type:	MQCFIL64.
Returned:	When available.

GetCount

Description:	The number of successful destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2
Identifier:	MQIAMO_GETS.
Data type:	MQCFIL.
Returned:	When available.

GetFailCount

Description:	The number of unsuccessful destructive get requests.
Identifier:	MQIAMO_GETS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

GetBytes

Description:	The number of bytes read in destructive gets requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2
Identifier:	MQIAMO64_GET_BYTES.
Data type:	MQCFIL64.
Returned:	When available.

BrowseCount

Description:	The number of successful non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2
Identifier:	MQIAMO_BROWSES.
Data type:	MQCFIL.
Returned:	When available.

BrowseFailCount

Description:	The number of unsuccessful non-destructive get requests.
Identifier:	MQIAMO_BROWSES_FAILED.
Data type:	MQCFIN.
Returned:	When available.

BrowseBytes

Description:	The number of bytes read in non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Reference note 2
Identifier:	MQIAMO64_BROWSE_BYTES.
Data type:	MQCFIL64.
Returned:	When available.

CommitCount

Description:	The number of transactions successfully completed. This number includes transactions committed implicitly by the application disconnecting, and commit requests where there is no outstanding work.
Identifier:	MQIAMO_COMMITS.
Data type:	MQCFIN.
Returned:	When available.

CommitFailCount

Description:	The number of unsuccessful attempts to complete a transaction.
Identifier:	MQIAMO_COMMITS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

BackCount

Description:	The number of backouts processed, including implicit backout upon abnormal disconnect.
Identifier:	MQIAMO_BACKOUTS.
Data type:	MQCFIN.
Returned:	When available.

ExpiredMsgCount

Description:	The number of persistent and nonpersistent messages that were discarded because they had expired, before they could be retrieved.
Identifier:	MQIAMO_MSGS_EXPIRED.
Data type:	MQCFIN.
Returned:	When available.

PurgeCount

Description:	The number of times the queue has been cleared.
Identifier:	MQIAMO_MSGS_PURGED.
Data type:	MQCFIN.
Returned:	When available.

SubCountDur

Description:	The number of successful Subscribe requests which created, altered or resumed durable subscriptions. This is an array of values indexed by the type of operation 0 = The number of subscriptions created 1 = The number of subscriptions altered 2 = The number of subscriptions resumed
Identifier:	MQIAMO_SUBS_DUR.
Data type:	MQCFIL
Returned:	When available.

SubCountNDur

Description:	The number of successful Subscribe requests which created, altered or resumed non-durable subscriptions. This is an array of values indexed by the type of operation 0 = The number of subscriptions created 1 = The number of subscriptions altered 2 = The number of subscriptions resumed
Identifier:	MQIAMO_SUBS_NDUR.
Data type:	MQCFIL.
Returned:	When available.

SubFailCount

Description:	The number of unsuccessful Subscribe requests.
Identifier:	MQIAMO_SUBS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

UnsubCountDur

Description:	The number of succesful unsubscribe requests for durable subscriptions. This is an array of values indexed by the type of operation 0 - The subscription was closed but not removed 1 - The subscription was closed and removed
Identifier:	MQIAMO_UNSUBS_DUR.
Data type:	MQCFIL.
Returned:	When available.

UnsubCountNDur

Description:	The number of succesful unsubscribe requests for non-durable subscriptions. This is an array of values indexed by the type of operation 0 - The subscription was closed but not removed 1 - The subscription was closed and removed
Identifier:	MQIAMO_UNSUBS_NDUR.
Data type:	MQCFIL.
Returned:	When available.

UnsubFailCount

Description:	The number of failed unsubscribe requests.
Identifier:	MQIAMO_UNSUBS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

SubRqCount

Description:	The number of successful MQSUBRQ requests.
Identifier:	MQIAMO_SUBRQS
Data type:	MQCFIN
Returned:	When available.

SubRqFailCount

Description:	The number of unsuccessful MQSUBRQ requests.
Identifier:	MQIAMO_SUBRQS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

CBCount

Description:	The number of successful MQCB requests. This is an array of values indexed by the type of operation 0 - A callback was created or altered 1 - A callback was removed 2 - A callback was resumed 3 - A callback was suspended
Identifier:	MQIAMO_CBS.
Data type:	MQCFIL.
Returned:	When available.

CBFailCount

Description:	The number of unsuccessful MQCB requests.
Identifier:	MQIAMO_CBS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

CtlCount

Description:	The number of successful MQCTL requests. This is an array of values indexed by the type of operation : 0 - The connection was started 1 - The connection was stopped 2 - The connection was resumed 3 - The connection was suspended
Identifier:	MQIAMO_CTLS.
Data type:	MQCFIL.
Returned:	When available.

CtlFailCount

Description:	The number of unsuccessful MQCTL requests.
Identifier:	MQIAMO_CTLS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

StatCount

Description:	The number of successful MQSTAT requests.
Identifier:	MQIAMO_STATS.
Data type:	MQCFIN.
Returned:	When available.

StatFailCount

Description:	The number of unsuccessful MQSTAT requests.
Identifier:	MQIAMO_STATS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

SubCountDurHighWater

Description:	The high-water mark on the number of durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE 0 - The high-water mark for all durable subscriptions in the system 1 - The high-water mark for durable application subscriptions (MQSUBTYPE_API) 2 - The high-water mark for durable admin subscription (MQSUBTYPE_ADMIN) 3 - The high-water mark for durable proxy subscriptions (MQSUBTYPE_PROXY)
Identifier:	MQIAMO_SUB_DUR_HIGHWATER
Data type:	MQCFIL.
Returned:	When available.

SubCountDurLowWater

Description:	The low-water mark on the number of durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE. 0 - The low-water mark for all durable subscriptions in the system 1 - The low-water mark for durable application subscriptions (MQSUBTYPE_API) 2 - The low-water mark for durable admin subscriptions (MQSUBTYPE_ADMIN) 3 - The low-water mark for durable proxy subscriptions (MQSUBTYPE_PROXY)
Identifier:	MQIAMO_SUB_DUR_LOWWATER
Data type:	MQCFIL.
Returned:	When available.

SubCountNDurHighWater

Description:	The high-water mark on the number of non-durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE 0 - The high-water mark for all non-durable subscriptions in the system 1 - The high-water mark for non-durable application subscriptions (MQSUBTYPE_API) 2 - The high-water mark for non-durable admin subscription (MQSUBTYPE_ADMIN) 3 - The high-water mark for non-durable proxy subscriptions (MQSUBTYPE_PROXY)
Identifier:	MQIAMO_SUB_NDUR_HIGHWATER
Data type:	MQCFIL.
Returned:	When available.

SubCountNDurLowWater

Description:	The low-water mark on the number of non-durable subscriptions during the time interval. This is an array of values indexed by SUBTYPE. 0 - The low-water mark for all non-durable subscriptions in the system 1 - The low-water mark for non-durable application subscriptions (MQSUBTYPE_API) 2 - The low-water mark for non-durable admin subscriptions (MQSUBTYPE_ADMIN) 3 - The low-water mark for non-durable proxy subscriptions (MQSUBTYPE_PROXY)
Identifier:	MQIAMO_SUB_NDUR_LOWWATER
Data type:	MQCFIL.
Returned:	When available.

PutTopicCount

Description:	The number persistent and nonpersistent messages successfully put to a topic, with the exception of messages put using the MQPUT1 call. This parameter is an integer list indexed by persistence value, see Reference note 2. Note: Messages put using a queue alias which resolve to a topic are included in this value.
Identifier:	MQIAMO_TOPIC_PUTS.
Data type:	MQCFIL.
Returned:	When available.

PutTopicFailCount

Description:	The number of unsuccessful attempts to put a message to a topic.
Identifier:	MQIAMO_TOPIC_PUTS_FAILED.
Data type:	MQCFIN.
Returned:	When available.

Put1TopicCount

Description:	The number of persistent and nonpersistent messages successfully put to a topic using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Reference note 2. Note: Messages put using a queue alias which resolve to a topic are included in this value.
Identifier:	MQIAMO_TOPIC_PUT1S.
Data type:	MQCFIL.
Returned:	When available.

Put1TopicFailCount

Description:	The number of unsuccessful attempts to put a message to a topic using MQPUT1 calls.
Identifier:	MQIAMO_TOPIC_PUT1S_FAILED.
Data type:	MQCFIN.
Returned:	When available.

PutTopicBytes

Description:	The number bytes written using put calls for persistent and nonpersistent messages which resolve to a publish operation. This is number of bytes put by the application and not the resultant number of bytes delivered to subscribers, see PublishMsgBytes for this value. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_TOPIC_PUT_BYTES.
Data type:	MQCFIL64.
Returned:	When available.

PublishMsgCount

Description:	The number of messages delivered to subscriptions in the time interval. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUBLISH_MSG_COUNT
Data type:	MQCFIL.
Returned:	When available.

PublishMsgBytes

Description:	The number of bytes delivered to subscriptions in the time interval. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_PUBLISH_MSG_BYTES
Data type:	MQCFIL64.
Returned:	When available.

Queue statistics message data:

Use this page to view the structure of a queue statistics message

Message name:	Queue statistics message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

QueueManager

Description:	Name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date at the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time at the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date at the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time at the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL
Data type:	MQCFIN
Returned:	Always

ObjectCount

Description:	The number of queue objects accessed in the interval for which statistics data has been recorded. This value is set to the number of QStatisticsData PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Data type:	MQCFIN
Returned:	Always

QStatisticsData

Description:	Grouped parameters specifying statistics details for a queue
Identifier:	MQGACF_Q_STATISTICS_DATA
Data type:	MQCFGR

Parameters in group:	*QName* *CreateDate* *CreateTime* *QType* *QDefinitionType* *QMinDepth* *QMaxDepth* *AvgTimeOnQ* *PutCount* *PutFailCount* *Put1Count* *Put1FailCount* *PutBytes* *GetCount* *GetFailCount* *GetBytes* *BrowseCount* *BrowseFailCount* *BrowseBytes* *NonQueuedMsgCount* *ExpiredMsgCount* *PurgeCount*
Returned:	Always

QName

Description:	The name of the queue
Identifier:	MQCA_Q_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always

CreateDate

Description:	The date when the queue was created
Identifier:	MQCA_CREATION_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

CreateTime

Description:	The time when the queue was created
Identifier:	MQCA_CREATION_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

QType

Description:	The type of the queue
Identifier:	MQIA_Q_TYPE
Data type:	MQCFIN
Value:	MQOT_LOCAL
Returned:	Always

QDefinitionType

Description:	The queue definition type
Identifier:	MQIA_DEFINITION_TYPE
Data type:	MQCFIN
Values:	Possible values are • MQQDT_PREDEFINED • MQQDT_PERMANENT_DYNAMIC • MQQDT_TEMPORARY_DYNAMIC
Returned:	When available

QMinDepth

Description:	The minimum queue depth during the monitoring period
Identifier:	MQIAMO_Q_MIN_DEPTH
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

QMaxDepth

Description:	The maximum queue depth during the monitoring period
Identifier:	MQIAMO_Q_MAX_DEPTH
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

AvgTimeOnQ

Description:	The average latency, in microseconds, of messages destructively retrieved from the queue during the monitoring period. This parameter is an integer list indexed by persistence value, see Reference note 2.
Identifier:	MQIAMO64_AVG_Q_TIME
Data type:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to the queue, with exception of MQPUT1 requests. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_PUTS
Data type:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message to the queue
Identifier:	MQIAMO_PUTS_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_PUT1S
Data type:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutBytes

Description:	The number of bytes written in put requests to the queue
Identifier:	MQIAMO64_PUT_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetCount

Description:	The number of successful destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_GETS
Data type:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

GeneratedMsgCount

Description:	The number of generated messages. Generated messages are: • Trigger messages • Queue Depth Low Events • Queue Depth Hi Events
Identifier:	MQIAMO_GENERATED_MSGS
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetFailCount

Description:	The number of unsuccessful destructive get requests
Identifier:	MQIAMO_GETS_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetBytes

Description:	The number of bytes read in destructive put requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO64_GET_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO_BROWSES
Data type:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive get requests
Identifier:	MQIAMO_BROWSES_FAILED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseBytes

Description:	The number of bytes read in non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Reference note 2.
Identifier:	MQIAMO64_BROWSE_BYTES
Data type:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

NonQueuedMsgCount

Description:	The number of messages that bypassed the queue and were transferred directly to a waiting application. Bypassing a queue can only occur in certain circumstances. This number represents how many times IBM MQ was able to bypass the queue, and not the number of times an application was waiting.
Identifier:	MQIAMO_MSGS_NOT_QUEUED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

ExpiredMsgCount

Description:	The number of persistent and nonpersistent messages that were discarded because they had expired before they could be retrieved.
Identifier:	MQIAMO_MSGS_EXPIRED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

PurgeCount

Description:	The number of messages purged.
Identifier:	MQIAMO_MSGS_PURGED
Data type:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

CBCount

Description:	The number of successful MQCB requests. This is an array of values indexed by the type of operation 0 - A callback was created or altered 1 - A callback was removed 2 - A callback was resumed 3 - A callback was suspended
Identifier:	MQIAMO_CBS
Data type:	MQCFIN
Returned:	When available.

CBFailCount

Description:	The number of unsuccessful MQCB requests.
Identifier:	MQIAMO_CBS_FAILED
Data type:	MQCFIN
Returned:	When available.

Channel statistics message data:

Use this page to view the structure of a channel statistics message

Message name:	Channel statistics message.
Platforms:	All, except IBM MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

QueueManager

Description:	The name of the queue manager.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

IntervalStartDate

Description:	The date at the start of the monitoring period.
Identifier:	MQCAMO_START_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

IntervalStartTime

Description:	The time at the start of the monitoring period.
Identifier:	MQCAMO_START_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH.
Returned:	Always.

IntervalEndDate

Description:	The date at the end of the monitoring period
Identifier:	MQCAMO_END_DATE.
Data type:	MQCFST.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

IntervalEndTime

Description:	The time at the end of the monitoring period
Identifier:	MQCAMO_END_TIME.
Data type:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

CommandLevel

Description:	The queue manager command level.
Identifier:	MQIA_COMMAND_LEVEL.
Data type:	MQCFIN.
Returned:	Always.

ObjectCount

Description:	The number of Channel objects accessed in the interval for which statistics data has been recorded. This value is set to the number of ChlStatisticsData PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Data type:	MQCFIN.
Returned:	Always.

ChlStatisticsData

Description:	Grouped parameters specifying statistics details for a channel.
Identifier:	MQGACF_CHL_STATISTICS_DATA.
Data type:	MQCFGR.
Parameters in group:	*ChannelName* *ChannelType* *RemoteQmgr* *ConnectionName* *MsgCount* *TotalBytes* *NetTimeMin* *NetTimeAvg* *NetTimeMax* *ExitTimeMin* *ExitTimeAvg* *ExitTimeMax* *FullBatchCount* *IncmplBatchCount* *AverageBatchSize* *PutRetryCount*
Returned:	Always.

ChannelName

Description:	The name of the channel.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

ChannelType

Description:	The channel type.
Identifier:	MQIACH_CHANNEL_TYPE.
Data type:	MQCFIN.
Values:	Possible values are: **MQCHT_SENDER** Sender channel. **MQCHT_SERVER** Server channel. **MQCHT_RECEIVER** Receiver channel. **MQCHT_REQUESTER** Requester channel. **MQCHT_CLUSRCVR** Cluster receiver channel. **MQCHT_CLUSSDR** Cluster sender channel.
Returned:	Always.

RemoteQmgr

Description:	The name of the remote queue manager.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	When available.

ConnectionName

Description:	Connection name of remote queue manager.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST
Maximum length:	MQ_CONN_NAME_LENGTH
Returned:	When available.

MsgCount

Description:	The number of persistent and nonpersistent messages sent or received.
Identifier:	MQIAMO_MSGS.
Data type:	MQCFIN
Returned:	When available.

TotalBytes

Description:	The number of bytes sent or received for persistent and nonpersistent messages.
Identifier:	MQIAMO64_BYTES.
Data type:	MQCFIN64.
Returned:	When available.

NetTimeMin

Description:	The shortest recorded channel round trip measured in the recording interval, in microseconds.
Identifier:	MQIAMO_NET_TIME_MIN.
Data type:	MQCFIN.
Returned:	When available.

NetTimeAvg

Description:	The average recorded channel round trip measured in the recording interval, in microseconds.
Identifier:	MQIAMO_NET_TIME_AVG.
Data type:	MQCFIN.
Returned:	When available.

NetTimeMax

Description:	The longest recorded channel round trip measured in the recording interval, in microseconds.
Identifier:	MQIAMO_NET_TIME_MAX.
Data type:	MQCFIN.
Returned:	When available.

ExitTimeMin

Description:	The shortest recorded time, in microseconds, spent executing a user exit in the recording interval,
Identifier:	MQIAMO_EXIT_TIME_MIN.
Data type:	MQCFIN.
Returned:	When available.

ExitTimeAvg

Description:	The average recorded time, in microseconds, spent executing a user exit in the recording interval. Measured in microseconds.
Identifier:	MQIAMO_EXIT_TIME_AVG.
Data type:	MQCFIN.
Returned:	When available.

ExitTimeMax

Description:	The longest recorded time, in microseconds, spent executing a user exit in the recording interval. Measured in microseconds.
Identifier:	MQIAMO_EXIT_TIME_MAX.
Data type:	MQCFIN.
Returned:	When available.

FullBatchCount

Description:	The number of batches processed by the channel that were sent because the value of the channel attributes BATCHSZ or BATCHLIM was reached.
Identifier:	MQIAMO_FULL_BATCHES.
Data type:	MQCFIN.
Returned:	When available.

IncmplBatchCount

Description:	The number of batches processed by the channel, that were sent without the value of the channel attribute BATCHSZ being reached.
Identifier:	MQIAMO_INCOMPLETE_BATCHES.
Data type:	MQCFIN.
Returned:	When available.

AverageBatchSize

Description:	The average batch size of batches processed by the channel.
Identifier:	MQIAMO_AVG_BATCH_SIZE.
Data type:	MQCFIN.
Returned:	When available.

PutRetryCount

Description:	The number of times in the time interval that a message failed to be put, and entered a retry loop.
Identifier:	MQIAMO_PUT_RETRIES.
Data type:	MQCFIN.
Returned:	When available.

Reference notes:

Use this page to view the notes to which descriptions of the structure of accounting and statistics messages refer

The following message data descriptions refer to these notes:
- "MQI accounting message data" on page 1080
- "Queue accounting message data" on page 1091
- "MQI statistics message data" on page 1101
- "Queue statistics message data" on page 1112
- "Channel statistics message data" on page 1119

1. This parameter relates to IBM MQ objects. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 124. Array indexed by object type

Object type	Value context
MQOT_Q (1)	Contains the value relating to queue objects.
MQOT_NAMELIST (2)	Contains the value relating to namelist objects.
MQOT_PROCESS (3)	Contains the value relating to process objects.
MQOT_Q_MGR (5)	Contains the value relating to queue manager objects.
MQOT_CHANNEL (6)	Contains the value relating to channel objects.
MQOT_AUTH_INFO (7)	Contains the value relating to authentication information objects.
MQOT_TOPIC (8)	Contains the value relating to topic objects.

   Note: An array of 13 MQCFIL or MQCFIL64 values are returned but only those listed are meaningful.

2. This parameter relates to IBM MQ messages. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 125. Array indexed by persistence value

Constant	Value
1	Contains the value for nonpersistent messages.
2	Contains the value for persistent messages.

Note: The index for each of these arrays starts at zero, so an index of 1 refers to the second row of the array. Elements of these arrays not listed in these tables contain no accounting or statistics information.

Application activity trace

Application activity trace produces detailed information about the behavior of applications connected to a queue manager. It traces the behavior of an application and provides a detailed view of the parameters used by an application as it interacts with IBM MQ resources. It also shows the sequence of MQI calls issued by an application.

Use Application activity trace when you require more information than is provided by Event monitoring, Message monitoring, Accounting and statistics messages, and Real-time monitoring.

Note that activity trace is not supported by IBM MQ for z/OS.

Collecting application activity trace information

An application activity trace message is a PCF message. You configure activity trace using a configuration file. To collect application activity trace information you set the ACTVTRC queue manager attribute. You can override this setting at connection level using MQCONNX options, or at application stanza level using the activity trace configuration file.

About this task

Activity trace messages are composed of an MQMD structure: a PCF (MQCFH) header structure, followed by a number of PCF parameters. A sequence of ApplicationTraceData PCF groups follows the PCF parameters. These PCF groups collect information about the MQI operations that an application performs while connected to a queue manager. You configure activity trace using a configuration file called `mqat.ini`.

To control whether or not application activity trace information is collected, you configure one or more of the following settings:

1. The ACTVTRC queue manager attribute.
2. The ACTVCONO settings (in the MQCNO structure passed in MQCONNX).
3. The matching stanza for the application in the activity trace configuration file `mqat.ini`.

The previous sequence is significant. The ACTVTRC attribute is overridden by the ACTVCONO settings, which are overridden by the settings in the `mqat.ini` file.

Trace entries are written after each operation has completed, unless otherwise stated. These entries are first written to the system queue SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE, then written to application activity trace messages when the application disconnects from the queue manager. For long running applications, intermediate messages are written if any of the following events occurs:

- The lifetime of the connection reaches a defined timeout value.
- The number of operations reaches a specified number.
- The amount of data collected in memory reaches the maximum message length allowed for the queue.

You set the timeout value using the `ActivityInterval` parameter. You set the number of operations using the `ActivityCount` parameter. Both parameters are specified in the activity trace configuration file `mqat.ini`.

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **ActivityCount** and the **ActivityInterval** settings. See "Tuning the performance impact of application activity trace" on page 1133.

The simplest way to view the contents of application activity trace messages is to use the "amqsact sample program" on page 1134.

Procedure

1. "Setting ACTVTRC to control collection of activity trace information."
2. "Setting MQCONNX options to control collection of activity trace information" on page 1127.
3. "Configuring activity trace behavior using `mqat.ini`" on page 1127.
4. "Tuning the performance impact of application activity trace" on page 1133.

Setting ACTVTRC to control collection of activity trace information:

Use the queue manager attribute ACTVTRC to control the collection of MQI application activity trace information

About this task

Application activity trace messages are generated only for connections that begin after application activity trace is enabled. The **ACTVTRC** parameter can have the following values:

ON API activity trace collection is switched on

OFF
API activity trace collection is switched off

Note: The **ACTVTRC** setting can be overridden by the queue manager **ACTVCONO** parameter. If you set the **ACTVCONO** parameter to `ENABLED`, then the **ACTVTRC** setting can be overridden for a given connection using the **Options** field in the MQCNO structure. See"Setting MQCONNX options to control collection of activity trace information" on page 1127.

Example

To change the value of the **ACTVTRC** parameter, you use the MQSC command `ALTER QMGR`. For example, to enable MQI application activity trace information collection use the following MQSC command:

```
ALTER QMGR ACTVTRC(ON)
```

What to do next

The simplest way to view the contents of application activity trace messages is to use the "amqsact sample program" on page 1134.

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **ActivityCount** and the **ActivityInterval** settings. See "Tuning the performance impact of application activity trace" on page 1133.

Setting MQCONNX options to control collection of activity trace information:

If the queue manager attribute **`ACTVCONO`** is set to `ENABLED`, you can use the **`ConnectOpts`** parameter on the MQCONNX call to enable or disable application activity reports on a per connection basis. These options override the activity trace behavior defined by the queue manager attribute **`ACTVTRC`**, and can be overridden by settings in the activity trace configuration file `mqat.ini`.

Procedure

1. Set the queue manager attribute **`ACTVCONO`** to `ENABLED`.

   Note: If an application attempts to modify the accounting behavior of an application using the **`ConnectOpts`** parameter, and the QMGR attribute **`ACTVCONO`** is set to `DISABLED`, then no error is returned to the application, and activity trace collection is defined by the queue manager attributes or the activity trace configuration file `mqat.ini`.
2. Set the **`ConnectOpts`** parameter on the MQCONNX call to `MQCNO_ ACTIVITY_ TRACE_ENABLED`.

   The **`ConnectOpts`** parameter on the MQCONNX call can have the following values:

   `MQCNO_ACTIVITY_ TRACE_DISABLED`
   Activity trace is switched off for the connection.

   `MQCNO_ ACTIVITY_ TRACE_ENABLED`
   Activity trace is switched on for the connection.

   Note: If an application selects both `MQCNO_ ACTIVITY_ TRACE_ENABLED` and `MQCNO_ACTIVITY_ TRACE_DISABLED` for MQCONNX, the call fails with a reason code of MQRC_OPTIONS_ERROR.
3. Check that these activity trace settings are not being overridden by settings in the activity trace configuration file `mqat.ini`.

   See"Configuring activity trace behavior using `mqat.ini`."

What to do next

The simplest way to view the contents of application activity trace messages is to use the "amqsact sample program" on page 1134.

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **`ActivityCount`** and the **`ActivityInterval`** settings. See "Tuning the performance impact of application activity trace" on page 1133.

Configuring activity trace behavior using `mqat.ini`:

Activity trace behavior is configured using a configuration file called `mqat.ini`. This file is used to define the level and frequency of reporting activity trace data. The file also provides a way to define rules to enable and disable activity trace based on the name of an application.

About this task

UNIX Linux On UNIX and Linux systems, `mqat.ini` is located in the queue manager data directory, which is the same location as the `qm.ini` file.

Windows On Windows systems, `mqat.ini` is located in the queue manager data directory `C:\Program Files\IBM\WebSphere MQ\qmgrs\queue_manager_name`. Users running applications to be traced need permission to read this file.

When the mqat.ini file is modified, newly created IBM MQ connections will be processed according to the modified version. Existing connections will continue to use the previous version unless the queue manager parameters are altered, for example following an ALTER QMGR command.

This file follows the same stanza key and parameter-value pair format as the mqs.ini and qm.ini files. It is located in the queue manager data directory, this is, the same location as the qm.ini file for the queue manager.

The file consists of a single stanza, **AllActivityTrace**, to configure the level and frequency of reporting activity trace data by default for all activity trace.

The file can also contain multiple **ApplicationTrace** stanzas. Each one of these, defines a rule for the trace behavior for one or more connections, based on matching the application name of the connections to the rule.

AllActivityTrace stanza

A single **AllActivityTrace** stanza defines settings for the activity trace that is applied to all IBM MQ connections, unless overridden.

Individual values in the **AllActivityTrace** stanza can be overridden by more specific information in an **ApplicationTrace stanza** stanza.

If more than one **AllActivityTrace** stanza is specified then the values in the last stanza is used. Parameters missing from the chosen **AllActivityTrace** take default values. Parameters and values from previous **AllActivityTrace** stanzas are ignored.

The following parameters can be specified under the **AllActivityTrace** stanza:

Table 126. Parameter/value pairs that can be used in the activity trace configuration file

Name	Values (default in bold type)	Description
ActivityInterval	0-99999999 (**1**)	Approximate time interval in seconds between trace messages. All activity performed by a connection in that interval will be written in a single message. If this value is 0, the trace message is written when the connection disconnects (or when the activity count is reached).
ActivityCount	0-99999999 (**100**)	Number of MQI or XA operations between trace messages. If this value is 0, the trace message is written when the connection disconnects (or when the activity interval has elapsed).
TraceLevel	LOW / **MEDIUM** / HIGH	Amount of parameter detail traced for each operation. The description of individual operations details which parameters are included for each trace level.
TraceMessageData	**0** - 104 857 600 (maximum 100 MB)	Amount of message data traced in bytes for MQGET, MQPUT, MQPUT1, and Callback operations
StopOnGetTraceMsg	**ON** / OFF	Using activity trace, to trace applications that are also processing activity trace messages, is not advisable due to possible looping occurring.

ApplicationTrace stanza

An ApplicationTrace stanza contains a rule which defines which IBM MQ connections will be traced or not trace based on the application name. Optionally, the default behaviour defined under the Allsettings which override the global trace level and frequency settings.

This stanza can include ApplName, ApplFunction and ApplClass parameters which are used according to the matching rules defined in Connection Matching Rules to determine whether the stanza applies to a particular connection or not.

The stanza must include the Trace parameter to determine if this rule turns activity trace on or off for matching connections.

An off rule can be used to explicitly disable trace for more specific application names and to override the ACTVTRC setting of the queue manager or activity trace connection options.

The following parameters can be specified under the **ApplicationTrace** stanza:

Table 127. Parameter/value pairs that can be used in the application trace configuration file

Name	**Values (default in bold type)**	**Description**
Trace	ON / OFF (Required parameter - no default value)	Activity trace switch. This switch can be used in the application-specific stanza to determine whether activity trace is active for the scope of the current application stanza. Note that this value overrides ACTVTRC and ACTVCONO settings for the queue manager.
ApplName	Character string (Required parameter - no default)	This value is used to determine which applications the ApplicationTrace stanza applies to. It is matched to the ApplName value from the API exit context structure (which is equivalent to the MQMD.PutApplName). The content of the ApplName value varies according to the application environment. For platforms other than z/OS, only the filename portion of the MQAXC.ApplName is matched to the value in the stanza. Characters to the left of the rightmost path separator are ignored when the comparison is made. z/OS For z/OS applications, the entire MQAXC.ApplName is matched to the value in the stanza. A single wildcard character (*) can be used at the end of the ApplName value to match any number of characters after that point. If the ApplName value is set to a single wildcard character (*) then the ApplName value matches all applications.

Table 127. Parameter/value pairs that can be used in the application trace configuration file (continued)

Name	**Values (default in bold type)**	**Description**
IBM i ApplFunction	Character string (default value *)	This value is used to qualify which application programs the **ApplicationTrace** stanza and **ApplName** value applies to. The stanza is optional, but is only valid for IBM i queue managers. A single wildcard character (*) can be used at the end of the **ApplName** value to match any number of characters. For example, an **ApplicationTrace** stanza specifying **ApplName** = * and **ApplFunction** = *AMQSPUT0* applies to all invocations of the AMQSPUT0 program from any job.
ApplClass	USER / MCA / **ALL**	The class of application. See the following table for an explanation of how the **AppType** values correspond to IBM MQ connections.

The following table shows how the *AppClass* values correspond to the **APICallerType** and **APIEnvironment** fields in the connection API exit context structure.

Table 128. Appclass values and how they correspond to the APICallerType and APIEnvironment fields

APPLCLASS	**API Caller Type:**	**API Environment:**	**Description**
USER	MQXACT_EXTERNAL	MQXE_OTHER	Only user applications are traced
MCA	(Any value)	MQXE_MCA MQXE_MCA_CLNTCONN MQXE_MCA_SVRCONN	Clients and channels (amqrmppa)
ALL	(Any value)	(Any value)	All connections are traced

Attention: You must use an **APPLCLASS** of *MCA* for client user applications, as a class of *USER* does not match these.

For example, to trace the **amqsputc** sample application, you could use the following code:

```
ApplicationTrace:
ApplClass=MCA                          # Application type
                                        #   Values:  (USER | MCA | INTERNAL | ALL)
                                        #   Default: USER
ApplName=amqsputc     # Application name (may be wildcarded)
                                        #   (matched to app name without path)
                                        #   Default: *
Trace=ON                                # Activity trace switch for application
                                        #   Values:  ( ON | OFF )
                                        #   Default: OFF
ActivityInterval=30                      # Time interval between trace messages
                                        #   Values: 0-99999999 (0=off)
                                        #   Default: 0
ActivityCount=1                         # Number of operations between trace msgs
                                        #   Values: 0-99999999  (0=off)
                                        #   Default: 0
TraceLevel=MEDIUM                       # Amount of data traced for each operation
                                        #   Values: LOW | MEDIUM | HIGH
                                        #   Default: MEDIUM
TraceMessageData=1000                   # Amount of message data traced
                                        #   Values: 0-100000000
                                        #   Default: 0
```

The default `mqat.ini` generated when a queue manager is created, contains a single rule to explicitly disable activity trace for the supplied activity trace sample, **amqsact**.

Connection Matching Rules

The queue manager applies the following rules to determine which stanzas settings to use for a connection.

1. A value specified in the **AllActivityTrace** stanza is used for the connection unless the value also occurs in an **ApplicationTrace** stanza and the stanza fulfills the matching criteria for the connection described in points 2, 3, and 4.
2. The **ApplClass** is matched against the type of the IBM MQ connection. If the **ApplClass** does not match the connection type then the stanza is ignored for this connection.
3. The *ApplName* value in the stanza is matched against the file name portion of the **ApplName** field from the API exit context structure (MQAXC) for the connection.

   The file name portion is derived from the characters to the right of the final path separator (/ or \) character. If the stanza **ApplName** includes a wildcard (*) then only the characters to the left of the wildcard are compared with the equivalent number of characters from the **ApplName** of the connection.

   For example, if a stanza value of "FRE*" is specified then only the first three characters are used in the comparison, so "path/FREEDOM" and "path\FREDDY" match, but "path/FRIEND" does not. If the *ApplName* value of the stanza does not match the connection **ApplName**, the stanza is ignored for this connection.
4. If more than one stanza matches the **ApplName** and **ApplClass** of the connection, then the stanza with the most specific **ApplName** is used.

   The most specific **ApplName** is defined as the one that uses the most characters to match the **ApplName** of the connection.

   For example, if the `ini` file contains a stanza with **ApplName** = *"FRE*"* and another stanza with **ApplName** = *"FREE*"* then the stanza with **ApplName** = *"FREE*"* is chosen as the best match for a connection with **ApplName** = *"path/FREEDOM"* because it matches four characters (whereas **ApplName** = *"FRE*"* matches only three).
5. If after applying the rules in points 2, 3, and 4, there is more than one stanza that matches the connections **ApplName** and **ApplClass** of the connection, the values from the last matching will be used and all other stanzas will be ignored.

Overriding default settings for each rule

Optionally, the global trace level and frequency settings under the **AllActivityTrace** stanza can be overridden for those connections matching an **ApplicationTrace** stanza.

The following parameters can be set under an **ApplicationTrace** stanza. If they are not set, the value is inherited from the **AllActivityTrace** stanza settings:

- **ActivityInterval**
- **ActivityCount**
- **TraceLevel**
- **TraceMessageData**
- **StopOnTraceMsg**

`mqat.ini` syntax

The syntax rules for the format of the `mqat.ini` file are:

- Text beginning with a hash or semicolon is considered to be a comment that extends to the end of the line.
- The first significant (non-comment) line must be a stanza key.
- A stanza key consists of the name of the stanza followed by a colon.

- A parameter-value pair consists of the name of a parameter followed by an equals sign and then the value.
- Only a single parameter-value pair can appear on a line. (A parameter-value must not wrap onto another line).
- Leading and trailing whitespace is ignored. There is no limit on the amount of white space between stanza names, parameter names and values, or parameter/value pairs. Line breaks are significant and not ignored
- The maximum length for any line is 2048 characters
- The stanza keys, parameter names, and constant parameter values are not case-sensitive, but the variable parameter values (*ApplName* and *DebugPath*) are case-sensitive.

Application Activity Trace File Example

The following example shows how the configuration data is specified in the Activity Trace ini file.

```
AllActivityTrace:
ActivityInterval=1
ActivityCount=100
TraceLevel=MEDIUM
TraceMessageData=0
StopOnGetTraceMsg=ON

ApplicationTrace:
ApplName=amqs*
Trace=ON
TraceLevel=HIGH
TraceMessageData=1000

ApplicationTrace:
ApplName=amqsact*
Trace=OFF
```

The above **AllActivityTrace** stanza defines how activity trace will perform by default when enabled, either through ApplicationTrace rules or through the queue manager ACTVTRC attribute or programmatically enabled by an application.

The first **ApplicationTrace** stanza defines a rule that will result in any MQI activity by an application whose name starts with "amqs" being traced. Trace generated for these applications will be of high detail and include up to 1000 bytes of message data. The activity interval and count parameters will be inherited

The second **ApplicationTrace** stanza defines a rule that turns trace off for applications with names starting "amqsact" (the activity trace sample). This rule will override the earlier 'on' rule for the amqsact application, resulting in no trace for that application.

An example is also shipped as a sample called `mqat.ini` in the C samples directory (the same directory as the amqsact.c file). This file can be copied to the queue manager data directory, for queue managers that have been migrated from an earlier release of IBM MQ.

What to do next

Enabling application activity trace can affect performance. The overhead can be reduced by tuning the **`ActivityCount`** and the **`ActivityInterval`** settings. See "Tuning the performance impact of application activity trace" on page 1133.

Tuning the performance impact of application activity trace:

Enabling application activity trace can incur a performance penalty. This can be reduced by only tracing the applications that you need, by increasing the number of applications draining the queue, and by tuning **`ActivityInterval`**, **`ActivityCount`** and **`TraceLevel`** in `mqat.ini`.

About this task

Enabling application activity trace selectively for an application or for all queue manager applications can result in additional messaging activity, and in the queue manager requiring additional storage space. In environments where messaging performance is critical, for example, in high workload applications or where a service level agreement (SLA) requires a minimum response time from the messaging provider, it might not be appropriate to collect application activity trace or it might be necessary to adjust the detail or frequency of trace activity messages that are produced. The preset values of **`ActivityInterval`**, **`ActivityCount`** and **`TraceLevel`** in the `mqat.ini` file give a default balance of detail and performance. However, you can tune these values to meet the precise functional and performance requirements of your system.

Procedure

- Only trace the applications that you need.

  Do this by creating an ApplicationTrace application-specific stanza in `mqat.ini`, or by changing the application to specify `MQCNO_ACTIVITY_TRACE_ENABLED` in the options field on the **`MQCNO`** structure on an MQCONNX call. See"Configuring activity trace behavior using `mqat.ini`" on page 1127and"Setting MQCONNX options to control collection of activity trace information" on page 1127.
- Before starting trace, check that at least one application is running and is ready to retrieve the activity trace message data from the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE.
- Keep the queue depth as low as possible, by increasing the number of applications draining the queue.
- Set the **`TraceLevel`** value in the `mqat.ini` file to collect the minimum amount of data required.

  `TraceLevel=LOW` has the lowest impact to messaging performance. See"Configuring activity trace behavior using `mqat.ini`" on page 1127.
- Tune the **`ActivityCount`** and **`ActivityInterval`** values in `mqat.ini`, to adjust how often activity trace messages are generated.

  If you are tracing multiple applications, the activity trace messages might be being produced faster than they can be removed from the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE. However, when you reduce how often activity trace messages are generated, you are also increasing the storage space required by the queue manager and the size of the messages when they are written to the queue.

What to do next

The simplest way to view the contents of application activity trace messages is to use the "amqsact sample program" on page 1134.

amqsact sample program

amqsact formats Application Activity Trace messages for you and is provided with IBM MQ.

The compiled program is located in the samples directory:

- On Linux and UNIX platforms `MQ_INSTALLATION_PATH/samp/bin`
- On Windows `MQ_INSTALLATION_PATH\tools\c\Samples\Bin`

Display mode

By default, **amqsact** in display mode processes messages on SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE. You can override this behavior by specifying a queue name or topic string.

You can also control the trace period displayed and specify whether the activity trace messages are removed or retained after display.

```
►►──amqsact── -m──QMgrName──┬──────────────┬─┬───────────────────┬─┬──────────────┬──────────────►
                            └─ -q──QName───┘ └─ -t──TopicString──┘ └─ -b──Browse──┘

►──┬──────────────────┬── -d──Depth── -w──Timeout── -s──StartTime── -e──EndTime──────────────────►◄
   └─ -v──Verbose─────┘
```

Required parameters

-m *QMgrName*
Name of the queue manager.

-d *Depth*
Number of records to display.

-w *Timeout*
Time to wait, in seconds. If no trace messages appear in the specified period, **amqsact** exits.

-s *StartTime*
Start time of record to process.

-e *EndTime*
End time of record to process.

Optional parameters

-q *QName*
Specify a specific queue to override the default queue name

-t *TopicString*
Subscribe to an event topic

-b Browse records only

-v Verbose output

Example output

Use **amqsact** on queue manager *TESTQM*, with verbose output, on an MQCONN API call:

```
amqsact -m TESTQM -v
```

The preceding command gives the following example output:

```
MonitoringType: MQI Activity Trace
Correl_id:
00000000: 414D 5143 5445 5354 514D 2020 2020 2020 'AMQCTESTQM      '
00000010: B5F6 4251 2000 E601                     '        '
```

```
QueueManager: 'TESTQM'
Host Name: 'ADMINIB-1VTJ6N1'
IntervalStartDate: '2014-03-15'
IntervalStartTime: '12:08:10'
IntervalEndDate: '2014-03-15'
IntervalEndTime: '12:08:10'
CommandLevel: 750
SeqNumber: 0
ApplicationName: 'IBM MQ_1\bin\amqsput.exe'
Application Type: MQAT_WINDOWS_7
ApplicationPid: 14076
UserId: 'Emma_Bushby'
API Caller Type: MQXACT_EXTERNAL
API Environment: MQXE_OTHER
Application Function: ''
Appl Function Type: MQFUN_TYPE_UNKNOWN
Trace Detail Level: 2
Trace Data Length: 0
Pointer size: 4
Platform: MQPL_WINDOWS_7
MQI Operation: 0
Operation Id: MQXF_CONN
ApplicationTid: 1
OperationDate: '2014-03-15'
OperationTime: '12:08:10'
ConnectionId:
00000000: 414D 5143 5445 5354 514D 2020 2020 2020 'AMQCTESTQM      '
00000010: FFFFFFB5FFFFFFF6 4251 2000 FFFFFE601  '         '
QueueManager: 'TESTQM'
Completion Code: MQCC_OK
Reason Code: 0
```

V 8.0.0.2

Dynamic mode

Dynamic mode applies only when using IBM MQ Appliance.

You enable dynamic mode by specifying an application name, a channel name, or a connection identifier as an argument to **amqsact**. Note that you can use wildcard characters in the name.

In dynamic mode, activity trace data is enabled at the start of the sample by use of a nondurable subscription to a system topic. Collecting activity trace data stops when **amqsact** stops. You must specify a timeout for **amqsact** in dynamic mode. You can run multiple copies of **amqsact** concurrently, with each instance receiving a copy of any activity trace data.

```
►►─amqsact── -m─QMgrName── -w─Timeout─┬─────────────────────┬─┬─────────────────┬──────►
                                      └─ -a─Application name─┘ └─ -c─Channel name─┘

►─┬──────────────────┬─┬─────────────┬────────────────────────────────────────────────►◄
  └─ -i─Connection ID─┘ └─ -v─Verbose─┘
```

V 8.0.0.2

Required parameters

-m *QMgrName*
    Name of the queue manager.

-w *Timeout*
    Time to wait, in seconds. If no trace messages appear in the specified period, **amqsact** exits.

V 8.0.0.2

Optional parameters

-a *Application name*
: Specify an application name to collect messages for

-c *Channel name*
: Specify a channel to collect messages for

-i *Connection ID*
: Specify a connection to collect messages for.

-v Verbose output

V 8.0.0.2

Example output

The following command generates and displays activity trace messages for any connections made by applications that start with the text "amqs". After 30 seconds of inactivity, the **amqsact** program ends, and no new activity trace data is generated.

```
amqsactc -m QMGR1 -w 30 -a amqs*
```

The following command generates and displays activity trace messages for any activity on the QMGR1.TO.QMGR2 channel. After 10 seconds of inactivity, the **amqsact** program ends, and no new activity trace data is generated.

```
amqsactc -m QMGR1 -w 10 -c QMGR1.TO.QMGR2
```

The following command generates and displays verbose activity trace messages for any activity on the existing IBM MQ connection that has a CONN of "6B576B5420000701", and an EXTCONN of "414D5143514D47523120202020202020". After a minute of inactivity, the **amqsact** program ends, and no new activity trace data is generated.

```
amqsactc -m QMGR1 -w 60 -i 414D5143514D475231202020202020206B576B5420000701 -v
```

Application activity trace message reference

Use this page to obtain an overview of the format of application activity trace messages and the information returned in these messages

Application activity trace messages are standard IBM MQ messages containing a message descriptor and message data. The message data contains information about the MQI operations performed by IBM MQ applications, or information about the activities occurring in an IBM MQ system.

Message descriptor
- An MQMD structure

Message data
- A PCF header (MQCFH)
- Application activity trace message data that is always returned
- Application activity trace message data that is operation-specific

Application activity trace message MQMD (message descriptor):

Use this page to understand the differences between the message descriptor of application activity trace messages and the message descriptor of event messages

The parameters and values in the message descriptor of application activity trace message are the same as in the message descriptor of event messages, with the following exception:

Format

Description:	Format name of message data.
Value:	**MQFMT_ADMIN** Admin message.

CorrelId

Description:	Correlation identifier.
Value:	Initialized with the ConnectionId of the application

MQCFH (PCF Header):

Use this page to view the PCF values contained by the MQCFH structure for an activity trace message

For an activity trace message, the MQCFH structure contains the following values:

Type

Description:	Structure type that identifies the content of the message.
Data type:	MQLONG.
Value:	MQCFT_APP_ACTIVITY

StrucLength

Description:	Length in bytes of MQCFH structure.
Data type:	MQLONG.
Value:	MQCFH_STRUC_LENGTH

Version

Description:	Structure version number.
Data type:	MQLONG.
Values:	MQCFH_VERSION_3

Command

Description:	Command identifier. This field identifies the category of the message.
Data type:	MQLONG.
Values:	MQCMD_ACTIVITY_TRACE

MsgSeqNumber

Description:	Message sequence number. This field is the sequence number of the message within a group of related messages.
Data type:	MQLONG.
Values:	1

Control

Description:	Control options.
Data type:	MQLONG.
Values:	MQCFC_LAST.

CompCode

Description:	Completion code.
Data type:	MQLONG.
Values:	MQCC_OK.

Reason

Description:	Reason code qualifying completion code.
Data type:	MQLONG.
Values:	MQRC_NONE.

ParameterCount

Description:	Count of parameter structures. This field is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Data type:	MQLONG.
Values:	1 or greater

Application activity trace message data:

Immediately following the PCF header is a set of parameters describing the time interval for the activity trace. These parameters also indicate the sequence of messages in the event of messages being written. The order and number of fields following the header is not guaranteed, allowing additional information to be added in the future.

Message name:	Activity trace message.
System queue:	SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE.

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

QSGName

z/OS

Description:	The name of QSG that the Queue Manager is a member of (z/OS only)
Identifier:	MQCA_QSG_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

HostName

Description:	The host name of the machine the Queue Manager is running on
Identifier:	MQCACF_HOST_NAME
Data type:	MQCFST

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Data type:	MQCFST
Maximum length:	MQ_DATE_LENGTH

IntervalEndTime

Description:	The time of the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Data type:	MQCFST
Maximum length:	MQ_TIME_LENGTH

CommandLevel

Description:	The IBM MQ command level
Identifier:	MQIA_COMMAND_LEVEL
Data type:	MQCFIN

SeqNumber

Description:	The sequence number normally zero. This value is incremented for each subsequent record for long running connections.
Identifier:	MQIACF_SEQUENCE_NUMBER
Data type:	MQCFIN

ApplicationName

Description:	The name of the application. (program name)
Identifier:	MQCACF_APPL_NAME
Data type:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH

ApplClass

Description:	Type of application that performed the activity. Possible values: MQAT_*
Identifier:	MQIA_APPL_TYPE
Data type:	MQCFIN

ApplicationPid

Description:	The operating system Process ID of the application.
Identifier:	MQIACF_PROCESS_ID
Data type:	MQCFIN

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Data type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH

APICallerType

Description:	The type of the application. Possible values: MQXACT_EXTERNAL or MQXACT_INTERNAL
Identifier:	MQIACF_API_CALLER_TYPE
Data type:	MQCFIN

Environment

Description:	The runtime environment of the application. Possible values: MQXE_OTHER MQXE_MCA MQXE_MCA_SVRCONN MQXE_COMMAND_SERVER MQXE_MQSC
Identifier:	MQIACF_API_ENVIRONMENT
Data type:	MQCFIN

Detail

Description:	The detail level that is recorded for the connection. Possible values: 1=LOW 2=MEDIUM 3=HIGH
Identifier:	MQIACF_TRACE_DETAIL
Data type:	MQCFIN

TraceDataLength

Description:	The length of message data (in bytes) that is traced for this connection.
Identifier:	MQIACF_TRACE_DATA_LENGTH
Data type:	MQCFIN

Pointer Size

Description:	The length (in bytes) of pointers on the platform the application is running (to assist in interpretation of binary structures)
Identifier:	MQIACF_POINTER_SIZE
Data type:	MQCFIN

Platform

Description:	The platform on which the queue manager is running. Value is one of the MQPL_* values.
Identifier:	MQIA_PLATFORM
Data type:	MQCFIN

Variable parameters for application activity MQI operations:

The application activity data `MQCFGR` structure is followed by the set of PCF parameters which corresponds to the operation being performed . The parameters for each operation are defined in the following section.

The trace level indicates the level of trace granularity that is required for the parameters to be included in the trace. The possible trace level values are:

1. Low

   The parameter is included when "low", "medium" or "high" activity tracing is configured for an application. This setting means that a parameter is always included in the AppActivityData group for the operation. This set of parameters is sufficient to trace the MQI calls an application makes, and to see if they are successful.
2. Medium

   The parameter is only included in the AppActivityData group for the operation when "medium" or "high" activity tracing is configured for an application. This set of parameters adds information about the resources, for example, queue and topic names used by the application.
3. High

   The parameter is only included in the AppActivityData group for the operation when "high" activity tracing is configured for an application. This set of parameters includes memory dumps of the structures passed to the MQI and XA functions. For this reason, it contains more information about the parameters used in MQI and XA calls. The structure memory dumps are shallow copies of the structures. To avoid erroneous attempts to dereference pointers, the pointer values in the structures are set to `NULL`.

   Note: The version of the structure that is dumped is not necessarily identical to the version used by an application. The structure can be modified by an API crossing exit, by the activity trace code, or by the queue manager. A queue manager can modify a structure to a later version, but the queue manager never changes it to an earlier version of the structure. To do so, would risk losing data.

MQBACK:

Application has started the MQBACK MQI function

`CompCode`

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

`Reason`

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

V 8.0.0.2 *`QMgrOpDuration`*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQBEGIN:

Application has started the MQBEGIN MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

MQBO

Description:	The MQBEGIN options structure. This parameter is not included if a NULL pointer is used on the MQBEGIN call.
PCF Parameter:	MQBACF_MQBO_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQBO structure.

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCALLBACK:

Application has started the MQCALLBACK function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type	MQCFIN

CallType

Description:	Why function has been called. One of the MQCBCT_* values
PCF Parameter:	MQIACF_CALL_TYPE
Trace level:	1
Type	MQCFIN

MsgBuffer

Description:	Message data.
PCF Parameter:	MQBACF_MESSAGE_DATA
Trace level:	1
Type	MQCFBS
Length:	Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. If TRACEDATA=NONE then this parameter is omitted.

MsgLength

Description:	Length of the message. (Taken from the DataLength field in the MQCBC structure).
PCF Parameter:	MQIACF_MSG_LENGTH
Trace level:	1
Type	MQCFIN

HighResTime

Description:	Time of operation in microseconds since midnight, January 1st 1970 (UTC) **Note:** The accuracy of this timer varies according to platform support for high a resolution timer
PCF Parameter:	MQIAMO64_HIGHRES_TIME
Trace level:	2
Type	MQCFIN64

ReportOptions

Description:	Options for report messages
PCF Parameter:	MQIACF_REPORT
Trace level:	2
Type	MQCFIN

MsgType

Description:	Type of message
PCF Parameter:	MQIACF_MSG_TYPE
Trace level:	2
Type	MQCFIN

Expiry

Description:	Message lifetime
PCF Parameter:	MQIACF_EXPIRY
Trace level:	2
Type	MQCFIN

Format

Description:	Format name of message data
PCF Parameter:	MQCACH_FORMAT_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_FORMAT_LENGTH

Priority

Description:	Message priority
PCF Parameter:	MQIACF_PRIORITY
Trace level:	2
Type	MQCFIN

Persistence

Description:	Message persistence
PCF Parameter:	MQIACF_PERSISTENCE
Trace level:	2
Type	MQCFIN

MsgId

Description:	Message identifier
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

ObjectName

Description:	The name of the opened object.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQName

Description:	The local name of the queue from which the message was retrieved.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH

ReplyToQueue

Description:	MQ_Q_NAME_LENGTH
PCF Parameter:	MQCACF_REPLY_TO_Q
Trace level:	2
Type	MQCFST

ReplyToQMgr

Description:	MQ_Q_MGR_NAME_LENGTH
PCF Parameter:	MQCACF_REPLY_TO_Q_MGR
Trace level:	2
Type	MQCFST

CodedCharSetId

Description:	Character set identifier of message data
PCF Parameter:	MQIA_CODED_CHAR_SET_ID
Trace level:	2
Type	MQCFIN

Encoding

Description:	Numeric encoding of message data.
PCF Parameter:	MQIACF_ENCODING
Trace level:	2
Type	MQCFIN

PutDate

Description:	MQ_PUT_DATE_LENGTH
PCF Parameter:	MQCACF_PUT_DATE
Trace level:	2
Type	MQCFST

PutTime

Description:	MQ_PUT_TIME_LENGTH
PCF Parameter:	MQCACF_PUT_TIME
Trace level:	2
Type	MQCFST

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL _Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description:	The put time of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_TIME
Trace level:	2
Type	MQCFST
Length:	MQ_PUT_TIME_LENGTH

XmitqPutDate

Description:	The put date of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_DATE
Trace level:	2
Type	MQCFST
Length:	MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description:	The remote queue destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_Name
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_MGR
Trace level:	2
Type	MQCFST
Length:	MQ_MSG_ID_LENGTH

MsgDescStructure

Description:	The MQMD structure. This parameter is omitted if a version 4 MQGMO was used to request that a Message Handle be returned instead of an MQMD
PCF Parameter:	MQBACF_MQMD_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQMD structure (actual size is dependent on structure version)

GetMsgOptsStructure

Description:	The MQGMO structure.
PCF Parameter:	MQBACF_MQGMO_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQGMO structure (actual size is dependent on structure version)

MQCBContextStructure

Description:	The MQCBC structure.
PCF Parameter:	MQBACF_MQCBC_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQCBC structure (actual size is dependent on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCB:

Application has started the manage callback MQI function

CallbackOperation

Description:	The manage callback function operation. Set to one of the MQOP_* values
PCF Parameter:	MQIACF_MQCB_OPERATION
Trace level:	1
Type	MQCFIN

CallbackType

Description:	The type of the callback function (CallbackType field from the MQCBD structure). Set to one of the MQCBT_* values
PCF Parameter:	MQIACF_MQCB_TYPE
Trace level:	1
Type	MQCFIN

CallbackOptions

Description:	The callback options. Set to one of the MQCBDO_* values
PCF Parameter:	MQIACF_MQCB_OPTIONS
Trace level:	1
Type	MQCFIN

CallbackFunction

Description:	The pointer to the callback function if started as a function call.
PCF Parameter:	MQBACF_MQCB_FUNCTION
Trace level:	1
Type	MQCFBS
Length:	Size of MQPTR

CallbackName

Description:	The name of the callback function if started as a dynamically linked program.
PCF Parameter:	MQCACF_MQCB_NAME
Trace level:	1
Type	MQCFST
Length:	Size of MQCHAR128

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type	MQCFIN

MaxMsgLength

Description:	Maximum message length. Set to an integer, or the special value MQCBD_FULL_MSG_LENGTH
PCF Parameter:	MQIACH_MAX_MSG_LENGTH
Trace level:	2
Type	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

CallBack DescriptorStructure

Description:	The MQCBD structure. This parameter is omitted if a NULL MQCBC value is passed to the MQCB call.
PCF Parameter:	MQBACF_MQCBD_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQCBC structure

MsgDescStructure

Description:	The MQMD structure. The MsgDescStructure parameter is omitted if a NULL MQMD value is passed to the MQCB call.
PCF Parameter:	MQBACF_MQMD_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQMD structure (actual size depends on structure version)

GetMsgOptsStructure

Description:	The MQGMO structure. This parameter is omitted if a NULL MQGMO value is passed to the MQCB call.
PCF Parameter:	MQBACF_MQGMO_STRUCT
Trace level:	3
Type	MQCFBS
Length:	The length in bytes of the MQGMO structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCLOSE:

Application has started the MQCLOSE MQI function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type	MQCFIN

CloseOptions

Description:	Close options
PCF Parameter:	MQIACF_CLOSE_OPTIONS
Trace level:	1
Type	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCMIT:

Application has started the MQCMIT MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type	MQCFIN

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCONN and MQCONNX:

Application has started the MQCONN or MQCONNX MQI function

ConnectionId

Description:	The Connection ID if available or MQCONNID_NONE if not
PCF Parameter:	MQBACF_CONNECTION_ID
Trace level:	1
Type:	MQCFBS
Maximum length:	MQ_CONNECTION_ID_LENGTH

QueueManagerName

Description:	The (unresolved) name of the queue manager used in the MQCONN(X) call
PCF Parameter:	MQCA_Q_MGR_NAME
Trace level:	1
Type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

ConnectOptions

Description:	Connect Options Derived from MQCNO_* values **Note:** MQCONNX only
PCF Parameter:	MQIACF_CONNECT_OPTIONS
Trace level:	2
Type:	MQCFIN

ConnectionOptionsStructure

Description:	The MQCNO structure. **Note:** MQCONNX only)
PCF Parameter:	MQBACF_MQCNO_STRUCT
Trace level:	3
Type:	MQCFBS
Maximum length:	The length in bytes of the MQCNO structure (actual size depends on structure version)

ChannelDefinitionStructure

Description:	The MQCD structure. **Note:** Client connections only
PCF Parameter:	MQBACF_MQCD_STRUCT
Trace level:	3
Type:	MQCFBS
Maximum length:	The length in bytes of the MQCD structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQCTL:

Application has started the MQCTL MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

CtlOperation

Description:	One of MQOP_* values
PCF Parameter:	MQIACF_CTL_OPERATION
Trace level:	1
Type:	MQCFIN

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQDISC:

Application has started the MQDISC MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

MQGET:

Application has started the MQGET MQI function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

GetOptions

Description:	The get options from MQGMO.Options
PCF Parameter:	MQIACF_GET_OPTIONS
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

MsgBuffer

Description:	Message data. If TRACEDATA=NONE then this parameter is omitted
PCF Parameter:	MQBACF_MESSAGE_DATA
Trace level:	1
Type:	MQCFBS
Maximum length:	Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. (Included in the trace message as MQIACF_TRACE_DATA_LENGTH).

MsgLength

Description:	Length of the message.
PCF Parameter:	MQIACF_MSG_LENGTH
Trace level:	1
Type:	MQCFIN

HighResTime

Description:	Time of operation in microseconds since midnight, January 1 1970 (UTC) **Note:** The accuracy of this timer varies according to platform support for high a resolution timer
PCF Parameter:	MQIAMO64_HIGHRES_TIME
Trace level:	2
Type:	MQCFIN64

BufferLength

Description:	Length of the buffer provided by the application
PCF Parameter:	MQIACF_BUFFER_LENGTH
Trace level:	2
Type:	MQCFIN

ObjectName

Description:	The name of the opened object
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQName

Description:	The local name of the queue from which the message was retrieved.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH

ReportOptions

Description:	Message report options
PCF Parameter:	MQIACF_REPORT
Trace level:	2
Type:	MQCFIN

MsgType

Description:	Type of message
PCF Parameter:	MQIACF_MSG_TYPE
Trace level:	2
Type:	MQCFIN

Expiry

Description:	Message lifetime
PCF Parameter:	MQIACF_EXPIRY
Trace level:	2
Type:	MQCFIN

Format

Description:	Format name of message data
PCF Parameter:	MQCACH_FORMAT_NAME
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_FORMAT_LENGTH

Priority

Description:	Message priority
PCF Parameter:	MQIACF_PRIORITY
Trace level:	2
Type:	MQCFIN

Persistence

Description:	Message persistence
PCF Parameter:	MQIACF_PERSISTENCE
Trace level:	2
Type:	MQCFIN

MsgId

Description:	Message identifier
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS
Maximum length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Maximum length:	MQ_CORREL_ID_LENGTH

ReplyToQueue

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH

ReplyToQMgr

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q_MGR
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

CodedCharSetId

Description:	Character set identifier of message data
PCF Parameter:	MQIA_CODED_CHAR_SET_ID
Trace level:	2
Type:	MQCFIN

Encoding

Description:	Numeric encoding of message data.
PCF Parameter:	MQIACF_ENCODING
Trace level:	2
Type:	MQCFIN

PutDate

Description:	
PCF Parameter:	MQCACF_PUT_DATE
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_PUT_DATE_LENGTH

PutTime

Description:	
PCF Parameter:	MQCACF_PUT_TIME
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_PUT_TIME_LENGTH

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description:	The put time of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

XmitqPutDate

Description:	The put date of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description:	The remote queue destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description:	The remote queue manager destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

MsgDescStructure

Description:	The MQMD structure.
PCF Parameter:	MQBACF_MQMD_STRUCT
Trace level:	3
Type:	MQCFBS
Maximum length:	The length in bytes of the MQMD structure (actual size depends on structure version)

GetMsgOptsStructure

Description:	The MQGMO structure.
PCF Parameter:	MQBACF_MQGMO_STRUCT
Trace level:	3
Type:	MQCFBS
Maximum length:	The length in bytes of the MQGMO structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQINQ:

Application has started the MQINQ MQI function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

SelectorCount

Description:	The count of selectors that are supplied in the Selectors array.
PCF Parameter:	MQIACF_SELECTOR_COUNT
Trace level:	2
Type:	MQCFIN

Selectors

Description:	The list of attributes (integer or character) whose values must be returned by MQINQ.
PCF Parameter:	MQIACF_SELECTORS
Trace level:	2
Type:	MQCFIL

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Maximum length:	Length varies

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type:	MQCFIN

IntAttrCount

Description:	The number of integer attributes returned by the inquire operation
PCF Parameter:	MQIACF_INTATTR_COUNT
Trace level:	3
Type:	MQCFIN

IntAttrs

Description:	The integer attribute values returned by the inquire operation. This parameter is only present if IntAttrCount is > 0 when MQINQ returns.
PCF Parameter:	MQIACF_INT_ATTRS
Trace level:	3
Type:	MQCFIL

CharAttrs

Description:	The character attributes returned by the inquire operation. The values are concatenated together. This parameter is only included if CharAttrLength is > 0 when MQINQ returns.
PCF Parameter:	MQCACF_CHAR_ATTRS
Trace level:	3
Type:	MQCFST

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQOPEN:

Application has started the MQOPEN MQI function

ObjectType

Description:	The object type passed in MQOT.ObjectType
PCF Parameter:	MQIACF_OBJECT_TYPE
Trace level:	1
Type:	MQCFIN

ObjectName

Description:	The name of the object passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	1
Type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH

ObjectQMgrName

Description:	The name of the object queue manager passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_Q_MGR_NAME
Trace level:	1
Type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

OpenOptions

Description:	Options used to open the object
PCF Parameter:	MQIACF_OPEN_OPTIONS
Trace level:	1
Type:	MQCFIN

AlternateUserId

Description:	Only included if MQOO_ALTERNATE_USER_AUTHORITY is specified
PCF Parameter:	MQCACF_ALTERNATE_USERID
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH

RecsPresent

Description:	The number of object name records present. Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_RECS_PRESENT
Trace level:	1
Type:	MQCFIN

KnownDestCount

Description:	Number of local queues opened successfully Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_KNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

UnknownDestCount

Description:	Number of remote queues opened successfully Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_UNKNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

InvalidDestCount

Description:	Number of queues that failed to open Only included if MQOD Version >= MQOD_VERSION_2
PCF Parameter:	MQIACF_INVALID_DEST_COUNT
Trace level:	1
Type:	MQCFIN

DynamicQName

Description:	The dynamic queue name passed as input to the MQOPEN call.
PCF Parameter:	MQCACF_DYNAMIC_Q_NAME
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH

ResolvedLocalQName [3] [4]

Description:	Contains the local queue name after name resolution has been carried out. (e.g. for remote queues this will be the name of the transmit queue)
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST
Range:	If MQOD.Version is less than MQOD_VERSION_3 this contains the value of the MQOD.ObjectName field after the MQOPEN call has completed. If MQOD.Version is equal or greater than MQOD_VERSION_3 this contains the value in the MQOD. ResolvedQName field.
Maximum length:	MQ_Q_NAME_LENGTH

ResolvedLocalQMgrName 3 4

Description:	The local queue manager name after name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_MGR
Trace level:	2
Type:	MQCFST
Range:	Only if MQOD.Version >= MQOD_VERSION_3
Maximum length:	MQ_Q_MGR_NAME_LENGTH

ResolvedQName 3 4

Description:	The queue name after name resolution has been carried out.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type:	MQCFST
Range:	If MQOD.Version is less than MQOD_VERSION_3 this contains the value of the MQOD.ObjectName field after the MQOPEN call has completed. If MQOD.Version is equal or greater than MQOD_VERSION_3 this contains the value in the MQOD. ResolvedQName field.
Maximum length:	MQ_Q_NAME_LENGTH

ResolvedQMgrName 3 4

Description:	Contains the queue manager name after name resolution has been carried out. If MQOD.Version is less than MQOD_VERSION_3 this contains the value of the MQOD. ObjectQMgrName field after the MQOPEN call has completed. If MQOD.Version is equal or greater than MQOD_VERSION_3 this contains the value in the MQOD. ResolvedQMgrName field.
PCF Parameter:	MQCACF_RESOLVED_Q_MGR
Trace level:	2
Type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH

AlternateSecurityId

Description:	Alternative security identifier. Only present if MQOD.Version is equal or greater than MQOD_VERSION_3, MQOO_ALTERNATE_USER_AUTHORITY is specified, and MQOD.AlternateSecurityId is not equal to MQSID_NONE.
PCF Parameter:	MQBACF_ALTERNATE_SECURITYID
Trace level:	2
Type:	MQCFBS
Maximum length:	MQ_SECURITY_ID_LENGTH

ObjectString

Description:	Long object name. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4 and the VSLength field of MQOD.ObjectString is MQVS_NULL_TERMINATED or greater than zero.
PCF Parameter:	MQCACF_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Maximum length:	Length varies.

SelectionString

Description:	Selection string. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4 and the VSLength field of MQOD. SelectionString is MQVS_NULL_TERMINATED or greater than zero.
PCF Parameter:	MQCACF_SELECTION_STRING
Trace level:	2
Type:	MQCFST
Maximum length:	Length varies.

ResObjectString

Description:	The long object name after the queue manager resolves the name provided in the ObjectName field. Only included for topics and queue aliases that reference a topic object if MQOD.Version is equal or greater than MQOD_VERSION_4 and VSLength is MQVS_NULL_TERMINATED or greater than zero.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Maximum length:	Length varies.

ResolvedType

Description:	The type of the resolved (base) object being opened. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type:	MQCFIN

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

Application Activity Distribution List PCF Group Header Structure:

If the MQOPEN function opens a distribution list, then the MQOPEN parameters includes one AppActivityDistList PCF group for each of the queues in the distribution list up to the number of structures numbered in RecsPresent. The Ap-pActivityDistList PCF group combines information from the MQOR, and MQRR structures to identify the queue name, and indicate the result of the open operation on the queue. An AppActivityDistList group always starts with the following MQCFGR structure:

Table 129. AppActivityDistList group MQCFGR structure

MQCFGR field	Value	Description
Type	MQCFT_GROUP	
StrucLength	Length in bytes of the MQCFGR structure	
Parameter	MQGACF_APP_DIST_LIST	Distribution list group parameter
ParameterCount	4	The number of parameter structures following the MQCFGR structure that are contained within this group.

ObjectName

Description:	The name of a queue in the distribution list MQ_Q_NAME_LENGTH. Only included if MQOR structures are provided.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH. Only included if MQOR structures are provided.

ObjectQMgrName

Description:	The name of the queue manager on which the queue named in ObjectName is defined.
PCF Parameter:	MQCACF_OBJECT_Q_MGR_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH. Only included if MQOR structures are provided.

CompCode

3. This parameter is only included if the object being opened resolves to a queue, and the queue is opened for MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_BROWSE

4. The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

Description:	The completion code indicating the result of the open for this object. Only included if MQRR structures are provided and the reason code for the MQOPEN is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	2
Type:	MQCFIN

Reason

Description:	The reason code indicating the result of the open for this object. Only included if MQRR structures are provided and the reason code for the MQOPEN is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	2
Type:	MQCFIN

MQPUT:

Application has started the MQPUT MQI function.

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

PutOptions

Description:	The put options from MQPMO.Options
PCF Parameter:	MQIACF_PUT_OPTIONS
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

MsgBuffer

Description:	Message data.
PCF Parameter:	MQBACF_MESSAGE_DATA
Trace level:	1
Type:	MQCFBS
Length:	Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. If TRACEDATA=NONE then this parameter is omitted.

MsgLength

Description:	Length of the message.
PCF Parameter:	MQIACF_MSG_LENGTH
Trace level:	1
Type:	MQCFIN

RecsPresent

Description:	The number of put message records or response records present. Only included if MQPMO Version >= MQPMO_VERSION_2
PCF Parameter:	MQIACF_RECS_PRESENT
Trace level:	1
Type:	MQCFIN

KnownDestCount

Description:	Number of messages sent successfully to local queues
PCF Parameter:	MQIACF_KNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

UnknownDestCount

Description:	Number of messages sent successfully to remote queues
PCF Parameter:	MQIACF_UNKNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

InvalidDestCount

Description:	Number of messages that could not be sent
PCF Parameter:	MQIACF_INVALID_DEST_COUNT
Trace level:	1
Type:	MQCFIN

HighResTime

Description:	Time of operation in microseconds since midnight, January 1st 1970 (UTC) **Note:** The accuracy of this timer varies according to platform support for high a resolution timer.
PCF Parameter:	MQIAMO64_HIGHRES_TIME
Trace level:	2
Type:	MQCFIN64

ObjectName

Description:	The name of the opened object.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQName

Description:	The name of the queue after queue name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQMgrName

Description:	The queue manager name after name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

ResolvedLocalQName [5]

Description:	Contains the local queue name after name resolution has been carried out.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST

ResolvedLocalQMgrName [5]

Description:	Contains the local queue manager name after name resolution has been carried out.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

ReportOptions

Description:	Message report options
PCF Parameter:	MQIACF_REPORT
Trace level:	2
Type:	MQCFIN

MsgType

Description:	Type of message
PCF Parameter:	MQIACF_MSG_TYPE
Trace level:	2
Type:	MQCFIN

Expiry

Description:	Message lifetime
PCF Parameter:	MQIACF_EXPIRY
Trace level:	2
Type:	MQCFIN

Format

Description:	Format name of message data
PCF Parameter:	MQCACH_FORMAT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_FORMAT_LENGTH

Priority

Description:	Message priority
PCF Parameter:	MQIACF_PRIORITY
Trace level:	2
Type:	MQCFIN

Persistence

Description:	Message persistence
PCF Parameter:	MQIACF_PERSISTENCE
Trace level:	2
Type:	MQCFIN

MsgId

Description:	Message identifier
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

ReplyToQueue

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ReplyToQMgr

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

CodedCharSetId

Description:	Character set identifier of message data
PCF Parameter:	MQIA_CODED_CHAR_SET_ID
Trace level:	2
Type:	MQCFIN

Encoding

Description:	Numeric encoding of message data.
PCF Parameter:	MQIACF_ENCODING
Trace level:	2
Type:	MQCFIN

PutDate

Description:	
PCF Parameter:	MQCACF_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

PutTime

Description:	
PCF Parameter:	MQCACF_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type:	MQCFIN

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description:	The put time of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

XmitqPutDate

Description:	The put date of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description:	The remote queue destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description:	The remote queue manager destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

PutMsgOptsStructure

Description:	The MQPMO structure.
PCF Parameter:	MQBACF_MQPMO_STRUCT
Trace level:	3
Type:	MQCFBS
Length:	The length in bytes of the MQPMO structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQPUT Application Activity Distribution List PCF Group Header Structure:

If the MQPUT function is putting to a distribution list, then the MQPUT parameters include one AppActivityDistList PCF group. For each of the queues in the distribution list, see "Application Activity Distribution List PCF Group Header Structure" on page 1170. The AppActivityDistList PCF group combines information from the MQPMR, and MQRR structures to identify the PUT parameters, and indicate the result of the PUT operation on each queue. For MQPUT operations the AppActivityDistList group contains some or all of the following parameters (the CompCode and Reason is present if the reason code is MQRC_MULTIPLE_REASONS and the other parameters are determined by the MQPMO.PutMsgRecFields field):

CompCode

Description:	The completion code indicating the result of the operation. Only included if MQRR structures are provided and the reason code for the MQPUT is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	2
Type:	MQCFIN

Reason

Description:	The reason code indicating the result of the put for this object. Only included if MQRR structures are provided and the reason code for the MQPUT is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	2
Type:	MQCFIN

MsgId

5. The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

Description:	Message identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_MSG_ID
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_CORREL_ID
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

GroupId

Description:	Group identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_GROUP_ID
PCF Parameter:	MQBACF_GROUP_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_GROUP_ID_LENGTH

Feedback

Description:	Feedback. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_FEEDBACK
PCF Parameter:	MQIACF_FEEDBACK
Trace level:	2
Type:	MQCFIN

AccountingToken

Description:	AccountingToken. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_ACCOUNTING_TOKEN
PCF Parameter:	MQBACF_ACCOUNTING_TOKEN
Trace level:	2
Type:	MQCFBS
Length:	MQ_ACCOUNTING_TOKEN_LENGTH.

MQPUT1:

Application has started the MQPUT1 MQI function

ObjectType

Description:	The object type passed in MQOT.ObjectType
PCF Parameter:	MQIACF_OBJECT_TYPE
Trace level:	1
Type:	MQCFIN

ObjectName

Description:	The name of the object passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	1
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ObjectQMgrName

Description:	The name of the object queue manager passed to the MQI call before any queue name resolution is attempted.
PCF Parameter:	MQCACF_OBJECT_Q_MGR_NAME
Trace level:	1
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

PutOptions

Description:	The put options from MQPMO.Options
PCF Parameter:	MQIACF_PUT_OPTIONS
Trace level:	1
Type:	MQCFIN

AlternateUserId

Description:	Only included if MQPMO_ALTERNATE_USER_AUTHORITY is specified.
PCF Parameter:	MQCACF_ALTERNATE_USERID
Trace level:	2
Type:	MQCFST
Length:	MQ_USER_ID_LENGTH

RecsPresent

Description:	The number of object name records present
PCF Parameter:	MQIACF_RECS_PRESENT
Trace level:	1
Type:	MQCFIN

KnownDestCount

Description:	Number of local queues opened successfully
PCF Parameter:	MQIACF_KNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

UnknownDestCount

Description:	Number of remote queues opened successfully
PCF Parameter:	MQIACF_UNKNOWN_DEST_COUNT
Trace level:	1
Type:	MQCFIN

InvalidDestCount

Description:	Number of queues that failed to open
PCF Parameter:	MQIACF_INVALID_DEST_COUNT
Trace level:	1
Type:	MQCFIN

MsgBuffer

Description:	Message data.
PCF Parameter:	MQBACF_MESSAGE_DATA
Trace level:	1
Type:	MQCFBS
Length:	Length is governed by the TRACEDATA() parameter set in the APPTRACE configuration. If TRACEDATA=NONE then this parameter is omitted.

MsgLength

Description:	Length of the message.
PCF Parameter:	MQIACF_MSG_LENGTH
Trace level:	1
Type:	MQCFIN

HighResTime

Description:	Time of operation in microseconds since midnight, January 1st 1970 (UTC) **Note:** The accuracy of this timer will vary according to platform support for high a resolution timer.
PCF Parameter:	MQIAMO64_HIGHRES_TIME
Trace level:	2
Type:	MQCFIN64

ResolvedQName

Description:	The name of the queue after queue name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ResolvedQMgrName

Description:	The queue manager name after name resolution has been performed.
PCF Parameter:	MQCACF_RESOLVED_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

ResolvedLocalQName [6]

Description:	Contains the local queue name after name resolution has been carried out
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type:	MQCFST

ResolvedLocalQMgrName [6]

Description:	Contains the local queue manager name after name resolution has been carried out.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_MGR_NAME_LENGTH

AlternateSecurityId

Description:	Alternate security identifier. Only present if MQOD.Version is equal or greater than MQOD_VERSION_3 and MQOD.AlternateSecurityId is not equal to MQSID_NONE.
PCF Parameter:	MQBACF_ALTERNATE_SECURITYID
Trace level:	2
Type:	MQCFBS
Length:	MQ_SECURITY_ID_LENGTH

ObjectString

Description:	Long object name. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4 and the VSLength field of MQOD.ObjectString is MQVS_NULL_TERMINATED or greater than zero.
PCF Parameter:	MQCACF_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Length:	Length varies.

ResObjectString

Description:	The long object name after the queue manager resolves the name provided in the ObjectName field. Only included for topics and queue aliases that reference a topic object if MQOD.Version is equal or greater than MQOD_VERSION_4 and VSLength is MQVS_NULL_TERMINATED or greater than zero.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type:	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the resolved (base) object being opened. Only included if MQOD.Version is equal or greater than MQOD_VERSION_4. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type:	MQCFIN

ReportOptions

Description:	Message report options
PCF Parameter:	MQIACF_REPORT
Trace level:	2
Type:	MQCFIN

MsgType

Description:	Type of message
PCF Parameter:	MQIACF_MSG_TYPE
Trace level:	2
Type:	MQCFIN

Expiry

Description:	Message lifetime
PCF Parameter:	MQIACF_EXPIRY
Trace level:	2
Type:	MQCFIN

Format

Description:	Format name of message data
PCF Parameter:	MQCACH_FORMAT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_FORMAT_LENGTH

Priority

Description:	Message priority
PCF Parameter:	MQIACF_PRIORITY
Trace level:	2
Type:	MQCFIN

Persistence

Description:	Message persistence
PCF Parameter:	MQIACF_PERSISTENCE
Trace level:	2
Type:	MQCFIN

MsgId

Description:	Message identifier
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

PCF Parameter:	Correlation identifier
Description:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

ReplyToQueue

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ReplyToQMgr

Description:	
PCF Parameter:	MQCACF_REPLY_TO_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQCFST

CodedCharSetId

Description:	Character set identifier of message data
PCF Parameter:	MQIA_CODED_CHAR_SET_ID
Trace level:	2
Type:	MQCFIN

Encoding

Description:	Numeric encoding of message data.
PCF Parameter:	MQIACF_ENCODING
Trace level:	2
Type:	MQCFIN

PutDate

Description:	
PCF Parameter:	MQCACF_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

PutTime

Description:	
PCF Parameter:	MQCACF_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

PolicyName

Description:	The policy name that was applied to this message. **Note:** AMS protected messages only
PCF Parameter:	MQCA_POLICY_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_OBJECT_NAME_LENGTH

XmitqMsgId

Description:	The message ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

XmitqCorrelId

Description:	The correlation ID of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQBACF_XQH_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

XmitqPutTime

Description:	The put time of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_TIME
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_TIME_LENGTH

XmitqPutDate

Description:	The put date of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_PUT_DATE
Trace level:	2
Type:	MQCFST
Length:	MQ_PUT_DATE_LENGTH

XmitqRemoteQName

Description:	The remote queue destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

XmitqRemoteQMgr

Description:	The remote queue manager destination of the message in the transmission queue header. **Note:** Only when Format is MQFMT_XMIT_Q_HEADER
PCF Parameter:	MQCACF_XQH_REMOTE_Q_MGR
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

PutMsgOptsStructure

Description:	The MQPMO structure.
PCF Parameter:	MQBACF_MQPMO_STRUCT
Trace level:	3
Type:	MQCFBS
Length:	The length in bytes of the MQPMO structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQPUT1 AppActivityDistList PCF Group Header Structure:

If the MQPUT1 function is putting to a distribution list, then the variable parameters include one AppActivityDistList PCF group. For each of the queues in the distribution list, see “Application Activity Distribution List PCF Group Header Structure” on page 1170. The AppActivityDistList PCF group combines information from the MQOR, MQPMR, and MQRR structures to identify the objects, and the PUT parameters , and indicate the result of the PUT operation on each queue. For MQPUT1 operations the AppActivityDistList group contains some or all of the following parameters (the CompCode, Reason, ObjectName, and ObjectQMgrName is present if the reason code is MQRC_MULTIPLE_REASONS and the other parameters is determined by the MQPMO.PutMsgRecFields field):

CompCode

6. The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

Description:	The completion code indicating the result of the put for this object. Only included if MQRR structures are provided and the reason code for the MQPUT1 is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	2
Type:	MQCFIN

Reason

Description:	The reason code indicating the result of the put for this object. Only included if MQRR structures are provided and the reason code for the MQPUT1 is MQRC_MULTIPLE_REASONS
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	2
Type:	MQCFIN

ObjectName

Description:	The name of a queue in the distribution list. Only included if MQOR structures are provided.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	2
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

MsgId

Description:	Message identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_MSG_ID
PCF Parameter:	MQBACF_MSG_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_MSG_ID_LENGTH

CorrelId

Description:	Correlation identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_CORREL_ID
PCF Parameter:	MQBACF_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

GroupId

Description:	Group identifier. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_GROUP_ID
PCF Parameter:	MQBACF_GROUP_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_GROUP_ID_LENGTH

Feedback

Description:	Feedback. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_FEEDBACK
PCF Parameter:	MQIACF_FEEDBACK
Trace level:	2
Type:	MQCFIN

AccountingToken

Description:	AccountingToken. Only included if MQPMR structures are provided.and PutMsgRecFields includes MQPMRF_ACCOUNTING_TOKEN
PCF Parameter:	MQBACF_ACCOUNTING_TOKEN
Trace level:	2
Type:	MQCFBS
Length:	MQ_ACCOUNTING_TOKEN_LENGTH.

MQSET:

Application has started the MQSET MQI function

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

SelectorCount

Description:	The count of selectors that are supplied in the Selectors array.
PCF Parameter:	MQIACF_SELECTOR_COUNT
Trace level:	2
Type:	MQCFIN

Selectors

Description:	The list of attributes (integer or character) whose values are being updated by MQSET.
PCF Parameter:	MQIACF_SELECTORS
Trace level:	2
Type:	MQCFIL

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

IntAttrCount

Description:	The number of integer attributes to be updated by the set operation.
PCF Parameter:	MQIACF_INTATTR_COUNT
Trace level:	3
Type:	MQCFIN

IntAttrs

Description:	The integer attribute values
PCF Parameter:	MQIACF_INT_ATTRS
Trace level:	3
Type:	MQCFIL
Range:	This parameter is only present if IntAttrCount is > 0

CharAttrs

Description:	The character attributes to be updated by the set operation. The values are concatenated together.
PCF Parameter:	MQCACF_CHAR_ATTRS
Trace level:	3
Type:	MQCFST
Range:	This parameter is only included if CharAttrLength is > 0

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQSUB:

Application has started the MQSUB MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

SubHandle

Description:	The subscription handle
PCF Parameter:	MQIACF_HSUB
Trace level:	1
Type:	MQCFIN

ObjectHandle

Description:	The object handle
PCF Parameter:	MQIACF_HOBJ
Trace level:	1
Type:	MQCFIN

Options

Description:	Subscription options
PCF Parameter:	MQIACF_SUB_OPTIONS
Trace level:	1
Type:	MQCFIN

ObjectName

Description:	The name of the object.
PCF Parameter:	MQCACF_OBJECT_NAME
Trace level:	1
Type:	MQCFST
Length:	MQ_Q_NAME_LENGTH

ObjectString

Description:	Long object name.
PCF Parameter:	MQCACF_OBJECT_STRING
Trace level:	1
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD.ObjectString is greater than zero or MQVS_NULL_TERMINATED.
Length:	Length varies.

AlternateUserId

Description:	
PCF Parameter:	MQCACF_ALTERNATE_USERID
Trace level:	2
Type:	MQCFST
Range:	Only included if MQSO_ALTERNATE_USER_AUTHORITY is specified.
Length:	MQ_USER_ID_LENGTH

AlternateSecurityId

Description:	Alternate security identifier.
PCF Parameter:	MQBACF_ALTERNATE_SECURITYID
Trace level:	2
Type:	MQCFBS
Range:	Only present if MQSO_ALTERNATE_USER_AUTHORITY is specified and MQSD.AlternateSecurityId is not equal to MQSID_NONE.
Length:	MQ_SECURITY_ID_LENGTH

SubName

Description:	Subscription Name
PCF Parameter:	MQCACF_SUB_NAME
Trace level:	2
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD.SubName is greater than zero or MQVS_NULL_TERMINATED.
Length:	Length varies.

SubUserData

Description:	Subscription User Data
PCF Parameter:	MQCACF_SUB_USER_DATA
Trace level:	2
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD.SubName is greater than zero or MQVS_NULL_TERMINATED.
Length:	Length varies.

SubCorrelId

Description:	Subscription Correlation identifier
PCF Parameter:	MQBACF_SUB_CORREL_ID
Trace level:	2
Type:	MQCFBS
Length:	MQ_CORREL_ID_LENGTH

SelectionString

Description:	Selection string.
PCF Parameter:	MQCACF_SELECTION_STRING
Trace level:	2
Type:	MQCFST
Range:	Only included if the VSLength field of MQSD. SelectionString is MQVS_NULL_TERMINATED or greater than zero.
Length:	Length varies.

ResolvedQName

Description:	The queue name referred to by the ObjectHandle, when ResolvedType is MQOT_Q.
PCF Parameter:	MQCACF_RESOLVED_LOCAL_Q_NAME
Trace level:	2
Type	MQCFST
Length:	MQ_Q_NAME_LENGTH.

ResObjectString

Description:	The object name referred to by the ObjectHandle, when ResolvedType is MQOT_TOPIC.
PCF Parameter:	MQCACF_RESOLVED_OBJECT_STRING
Trace level:	2
Type	MQCFST
Length:	Length varies.

ResolvedType

Description:	The type of the object referred to by the ObjectHandle. Possible values are MQOT_Q, MQOT_TOPIC, or MQOT_NONE.
PCF Parameter:	MQIACF_RESOLVED_TYPE
Trace level:	2
Type	MQCFIN

SubDescriptorStructure

Description:	The MQSD structure.
PCF Parameter:	MQBACF_MQSD_STRUCT
Trace level:	3
Type:	MQCFBS
Length:	The length in bytes of the MQSD structure.

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQSUBRQ:

Application has started the MQSUBRQ MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

SubHandle

Description:	The subscription handle
PCF Parameter:	MQIACF_HSUB
Trace level:	1
Type:	MQCFIN

SubOptions

Description:	The sub options from MQSB.Options
PCF Parameter:	MQIACF_SUBRQ_OPTIONS
Trace level:	2
Type:	MQCFIN

Action

Description:	The subscription request action (MQSR_*)
PCF Parameter:	MQIACF_SUBRQ_ACTION
Trace level:	2
Type:	MQCFIN

NumPubs

Description:	The number of publications sent as a result of this call (from MQSB.NumPubs)
PCF Parameter:	MQIACF_NUM_PUBS
Trace level:	2
Type:	MQCFIN

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

MQSTAT:

Application has started the MQSTAT MQI function

CompCode

Description:	The completion code indicating the result of the operation
PCF Parameter:	MQIACF_COMP_CODE
Trace level:	1
Type:	MQCFIN

Reason

Description:	The reason code result of the operation
PCF Parameter:	MQIACF_REASON_CODE
Trace level:	1
Type:	MQCFIN

Type

Description:	Type of status information being requested
PCF Parameter:	MQIACF_STATUS_TYPE
Trace level:	2
Type:	MQCFIN

StatusStructure

Description:	The MQSTS structure.
PCF Parameter:	MQBACF_MQSTS_STRUCT
Trace level:	3
Type:	MQCFBS
Length:	The length in bytes of the MQSTS structure (actual size depends on structure version)

V 8.0.0.2 *QMgrOpDuration*

Description:	Approximate API call duration, in microseconds, within the queue manager. The duration does not include the time spent outside of the queue manager. For example, the time taken as an IBM MQ client. **Note:** The accuracy of this timer varies according to the platform that your enterprise uses.
PCF Parameter:	MQIAMO64_QMGR_OP_DURATION
Trace level:	2
Type	MQCFIN64

Variable Parameters for Application Activity XA Operations:

XA operations are API calls that applications can make to enable MQ to participate in a transaction. The parameters for each operation are defined in the following section.

The trace level indicates the level of trace granularity that is required for the parameters to be included in the trace. The possible trace level values are:

1. Low

   The parameter is included when “low”, “medium” or “high” activity tracing is configured for an application. This setting means that a parameter is always included in the AppActivityData group for the operation. This set of parameters is sufficient to trace the MQI calls an application makes, and to see if they are successful.
2. Medium

   The parameter is only included in the AppActivityData group for the operation when “medium” or “high” activity tracing is configured for an application. This set of parameters adds information about the resources, for example, queue and topic names used by the application.
3. High

   The parameter is only included in the AppActivityData group for the operation when “high” activity tracing is configured for an application. This set of parameters includes memory dumps of the structures passed to the MQI and XA functions. For this reason, it contains more information about the parameters used in MQI and XA calls. The structure memory dumps are shallow copies of the structures. To avoid erroneous attempts to dereference pointers, the pointer values in the structures are set to `NULL`.

   Note: The version of the structure that is dumped is not necessarily identical to the version used by an application. The structure can be modified by an API crossing exit, by the activity trace code, or by the queue manager. A queue manager can modify a structure to a later version, but the queue manager never changes it to an earlier version of the structure. To do so, would risk losing data.

AXREG:

Application has started the AXREG AX function

`XID`

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

`Rmid`

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

`Flags`

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

AXUNREG:

Application has started the AXUNREG AX function

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XACLOSE:

Application has started the XACLOSE AX function

Xa_info

Description:	Information used to initialize the resource manager.
PCF Parameter:	MQCACF_XA_INFO
Trace level:	1
Type:	MQCFST

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XACOMMIT:

Application has started the XACOMMIT AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XACOMPLETE:

Application has started the XACOMPLETE AX function

Handle

Description:	Handle to async operation
PCF Parameter:	MQIACF_XA_HANDLE
Trace level:	1
Type:	MQCFIN

Retval

Description:	Return value of the asynchronous function
PCF Parameter:	MQIACF_XA_RETVAL
Trace level:	1
Type:	MQCFINMQCFBS

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAEND:

Application has started the XAEND AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAFORGET:

Application has started the AXREG AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAOPEN:

Application has started the XAOPEN AX function

Xa_info

Description:	Information used to initialize the resource manager.
PCF Parameter:	MQCACF_XA_INFO
Trace level:	1
Type:	MQCFST

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAPREPARE:

Application has started the XAPREPARE AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XARECOVER:

Application has started the XARECOVER AX function

Count

Description:	Count of XIDs
PCF Parameter:	MQIACF_XA_COUNT
Trace level:	1
Type:	MQCFIN

XIDs

Description:	The XID structures **Note:** There are multiple instances of this PCF parameter - one for every XID structure up to Count XIDs
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XAROLLBACK:

Application has started the XAROLLBACK AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

XASTART:

Application has started the XASTART AX function

XID

Description:	The XID structure
PCF Parameter:	MQBACF_XA_XID
Trace level:	1
Type:	MQCFBS
Length:	Sizeof(XID)

Rmid

Description:	Resource manager identifier
PCF Parameter:	MQIACF_XA_RMID
Trace level:	1
Type:	MQCFIN

Flags

Description:	Flags
PCF Parameter:	MQIACF_XA_FLAGS
Trace level:	1
Type:	MQCFIN

XARetCode

Description:	Return code
PCF Parameter:	MQIACF_XA_RETCODE
Trace level:	1
Type:	MQCFIN

Real-time monitoring

Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. The information returned is accurate at the moment the command was issued.

A number of commands are available that when issued return real-time information about queues and channels. Information can be returned for one or more queues or channels and can vary in quantity. Real-time monitoring can be used in the following tasks:

- Helping system administrators understand the steady state of their IBM MQ system. This helps with problem diagnosis if a problem occurs in the system.
- Determining the condition of your queue manager at any moment, even if no specific event or problem has been detected.
- Assisting with determining the cause of a problem in your system.

With real-time monitoring, information can be returned for either queues or channels. The amount of real-time information returned is controlled by queue manager, queue, and channel attributes.

- You monitor a queue by issuing commands to ensure that the queue is being serviced properly. Before you can use some of the queue attributes, you must enable them for real-time monitoring.
- You monitor a channel by issuing commands to ensure that the channel is running properly. Before you can use some of the channel attributes, you must enable them for real-time monitoring.

Real-time monitoring for queues and channels is in addition to, and separate from, performance and channel event monitoring.

Attributes that control real-time monitoring

Some queue and channel status attributes hold monitoring information, if real-time monitoring is enabled. If real-time monitoring is not enabled, no monitoring information is held in these monitoring attributes. Examples demonstrate how you can use these queue and channel status attributes.

You can enable or disable real-time monitoring for individual queues or channels, or for multiple queues or channels. To control individual queues or channels, set the queue attribute MONQ or the channel attribute MONCHL, to enable or disable real-time monitoring. To control many queues or channels together, enable or disable real-time monitoring at the queue manager level by using the queue manager attributes MONQ and MONCHL. For all queue and channel objects with a monitoring attribute that is specified with the default value, QMGR, real-time monitoring is controlled at the queue manager level.

Automatically defined cluster-sender channels are not IBM MQ objects, so do not have attributes in the same way as channel objects. To control automatically defined cluster-sender channels, use the queue manager attribute, MONACLS. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel monitoring.

For real-time monitoring of channels, you can set the MONCHL attribute to one of the three monitoring levels: low, medium, or high. You can set the monitoring level either at the object level or at the queue manager level. The choice of level is dependent on your system. Collecting monitoring data might require some instructions that are relatively expensive computationally, such as obtaining system time. To reduce the effect of real-time monitoring, the medium and low monitoring options measure a sample of the data at regular intervals rather than collecting data all the time. Table 130 summarizes the monitoring levels available for real-time monitoring of channels:

Table 130. Monitoring levels

Level	Description	Usage
Low	Measure a small sample of the data, at regular intervals.	For objects that process a high volume of messages.
Medium	Measure a sample of the data, at regular intervals.	For most objects.
High	Measure all data, at regular intervals.	For objects that process only a few messages per second, on which the most current information is important.

For real-time monitoring of queues, you can set the MONQ attribute to one of the three monitoring levels, low, medium or high. However, there is no distinction between these values. The values all enable data collection, but do not affect the size of the sample.

Examples

The following examples demonstrate how to set the necessary queue, channel, and queue manager attributes to control the level of monitoring. For all of the examples, when monitoring is enabled, queue and channel objects have a medium level of monitoring.

1. To enable both queue and channel monitoring for all queues and channels at the queue manager level, use the following commands:
   ```
   ALTER QMGR MONQ(MEDIUM) MONCHL(MEDIUM)
   ALTER QL(Q1) MONQ(QMGR)
   ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(QMGR)
   ```
2. To enable monitoring for all queues and channels, with the exception of local queue, `Q1`, and sender channel, `QM1.TO.QM2`, use the following commands:
   ```
   ALTER QMGR MONQ(MEDIUM) MONCHL(MEDIUM)
   ALTER QL(Q1) MONQ(OFF)
   ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(OFF)
   ```

3. To disable both queue and channel monitoring for all queues and channels, with the exception of local queue, `Q1`, and sender channel, `QM1.TO.QM2`, use the following commands:

```
ALTER QMGR MONQ(OFF) MONCHL(OFF)
ALTER QL(Q1) MONQ(MEDIUM)
ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(MEDIUM)
```

4. To disable both queue and channel monitoring for all queues and channels, regardless of individual object attributes, use the following command:

```
ALTER QMGR MONQ(NONE) MONCHL(NONE)
```

5. To control the monitoring capabilities of automatically defined cluster-sender channels use the following command:

```
ALTER QMGR MONACLS(MEDIUM)
```

6. To specify that automatically defined cluster-sender channels are to use the queue manager setting for channel monitoring, use the following command:

```
ALTER QMGR MONACLS(QMGR)
```

Related concepts:

"Real-time monitoring" on page 1206
Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. The information returned is accurate at the moment the command was issued.

"Using IBM MQ online monitoring" on page 1223
You can collect monitoring data for queues and channels (including automatically defined cluster-server channels) by setting the MONQ, MONCHL, and MONACLS attributes.

Related tasks:

"Displaying queue and channel monitoring data"
To display real-time monitoring information for a queue or channel, use either the IBM MQ Explorer or the appropriate MQSC command. Some monitoring fields display a comma-separated pair of indicator values, which help you to monitor the operation of your queue manager. Examples demonstrate how you can display monitoring data.

Related information:

Working with queue managers

Monitoring (MONCHL)

Displaying queue and channel monitoring data

To display real-time monitoring information for a queue or channel, use either the IBM MQ Explorer or the appropriate MQSC command. Some monitoring fields display a comma-separated pair of indicator values, which help you to monitor the operation of your queue manager. Examples demonstrate how you can display monitoring data.

About this task

Monitoring fields that display a pair of values separated by a comma provide short term and long term indicators for the time measured since monitoring was enabled for the object, or from when the queue manager was started:

- The short term indicator is the first value in the pair and is calculated in a way such that more recent measurements are given a higher weighting and will have a greater effect on this value. This gives an indication of recent trend in measurements taken.
- The long term indicator in the second value in the pair and is calculated in a way such that more recent measurements are not given such a high weighting. This gives an indication of the longer term activity on performance of a resource.

These indicator values are most useful to detect changes in the operation of your queue manager. This requires knowledge of the times these indicators show when in normal use, in order to detect increases in these times. By collecting and checking these values regularly you can detect fluctuations in the operation of your queue manager. This can indicate a change in performance.

Obtain real-time monitoring information as follows:

Procedure

1. To display real-time monitoring information for a queue, use either the IBM MQ Explorer or the MQSC command `DISPLAY QSTATUS`, specifying the optional parameter `MONITOR`.
2. To display real-time monitoring information for a channel, use either the IBM MQ Explorer or the MQSC command `DISPLAY CHSTATUS`, specifying the optional parameter `MONITOR`.

Example

The queue, `Q1`, has the attribute MONQ set to the default value, QMGR, and the queue manager that owns the queue has the attribute MONQ set to MEDIUM. To display the monitoring fields collected for this queue, use the following command:

```
DISPLAY QSTATUS(Q1) MONITOR
```

The monitoring fields and monitoring level of queue, `Q1` are displayed as follows:

```
QSTATUS(Q1)
TYPE(QUEUE)
MONQ(MEDIUM)
QTIME(11892157,24052785)
MSGAGE(37)
LPUTDATE(2005-03-02)
LPUTTIME(09.52.13)
LGETDATE(2005-03-02)
LGETTIME(09.51.02)
```

The sender channel, `QM1.TO.QM2`, has the attribute MONCHL set to the default value, QMGR, and the queue manager that owns the queue has the attribute MONCHL set to MEDIUM. To display the monitoring fields collected for this sender channel, use the following command:

```
DISPLAY CHSTATUS(QM1.TO.QM2) MONITOR
```

The monitoring fields and monitoring level of sender channel, `QM1.TO.QM2` are displayed as follows:

```
CHSTATUS(QM1.TO.QM2)
XMITQ(Q1)
CONNAME(127.0.0.1)
CURRENT
CHLTYPE(SDR)
STATUS(RUNNING)
SUBSTATE(MQGET)
MONCHL(MEDIUM)
XQTIME(755394737,755199260)
NETTIME(13372,13372)
EXITTIME(0,0)
XBATCHSZ(50,50)
COMPTIME(0,0)
STOPREQ(NO)
RQMNAME(QM2)
```

Related concepts:

"Real-time monitoring" on page 1206
Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. The information returned is accurate at the moment the command was issued.

Related information:

DISPLAY QSTATUS

Monitoring queues

Use this page to view tasks that help you to resolve a problem with a queue and the application that services that queue. Various monitoring options are available to determine the problem

Frequently, the first sign of a problem with a queue that is being serviced is that the number of messages on the queue (CURDEPTH) increases. If you expect an increase at certain times of day or under certain workloads, an increasing number of messages might not indicate a problem. However, if you have no explanation for the increasing number of messages, you might want to investigate the cause.

You might have an application queue where there is a problem with the application, or a transmission queue where there is a problem with the channel. Additional monitoring options are available when the application that services the queue is a channel.

The following examples investigate problems with a particular queue, called Q1, and describe the fields that you look at in the output of various commands:

Determining whether your application has the queue open:

If you have a problem with a queue, check whether your application has the queue open

About this task

Perform the following steps to determine whether your application has the queue open:

Procedure

1. Ensure that the application that is running against the queue is the application that you expect. Issue the following command for the queue in question:
   ```
   DISPLAY QSTATUS(Q1) TYPE(HANDLE) ALL
   ```
   In the output, look at the APPLTAG field, and check that the name of your application is shown. If the name of your application is not shown, or if there is no output at all, start your application.
2. If the queue is a transmission queue, look in the output at the CHANNEL field. If the channel name is not shown in the CHANNEL field, determine whether the channel is running.
3. Ensure that the application that is running against the queue has the queue open for input. Issue the following command:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   In the output, look at the IPPROCS field to see if any application has the queue open for input. If the value is 0 and this is a user application queue, make sure that the application opens the queue for input to get the messages off the queue.

Checking that messages on the queue are available:

If you have a large number of messages on the queue and your application is not processing any of those messages, check whether the messages on the queue are available to your application

About this task

Perform the following steps to investigate why your application is not processing messages from the queue:

Procedure

1. Ensure that your application is not asking for a specific message ID or correlation ID when it should be processing all the messages on the queue.
2. Although the current depth of the queue might show that there is an increasing number of messages on the queue, some messages on the queue might not be available to be got by an application, because they are not committed; the current depth includes the number of uncommitted MQPUTs of messages to the queue. Issue the following command:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   In the output, look at the UNCOM field to see whether there are any uncommitted messages on the queue.
3. If your application is attempting to get any messages from the queue, check whether the putting application is committing the messages correctly. Issue the following command to find out the names of applications that are putting messages to this queue:
   ```
   DISPLAY QSTATUS(Q1) TYPE(HANDLE) OPENTYPE(OUTPUT)
   ```
4. Then issue the following command, inserting in <appltag> the APPLTAG value from the output of the previous command:
   ```
   DISPLAY CONN(*) WHERE(APPLTAG EQ <appltag>) UOWSTDA UOWSTTI
   ```
   This shows when the unit of work was started and will help you discover whether the application is creating a long running unit of work. If the putting application is a channel, you might want to investigate why a batch is taking a long time to complete.

Checking whether your application is getting messages off the queue:

If you have a problem with a queue and the application that services that queue, check whether your application is getting messages off the queue

About this task

To check whether your application is getting messages off the queue, perform the following checks:

Procedure

1. Ensure that the application that is running against the queue is actually processing messages from the queue. Issue the following command:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   In the output, look at the LGETDATE and LGETTIME fields which show when the last get was done from the queue.
2. If the last get from this queue was longer ago than expected, ensure that the application is processing messages correctly. If the application is a channel, check whether messages are moving through that channel

Determining whether the application can process messages fast enough:

If messages are building up on the queue, but your other checks have not found any processing problems, check that the application can process messages fast enough. If the application is a channel, check that the channel can process messages fast enough.

About this task

To determine whether the application is processing messages fast enough, perform the following tests:

Procedure

1. Issue the following command periodically to gather performance data about the queue:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   If the values in the QTIME indicators are high, or are increasing over the period, and you have already ruled out the possibility of long running Units of Work by checking that messages on the queue are available, the getting application might not be keeping up with the putting applications.
2. If your getting application cannot keep up with the putting applications, consider adding another getting application to process the queue. Whether you can add another getting application depends on the design of the application and whether the queue can be shared by more than one application. Features such as message grouping or getting by correlation ID might help to ensure that two applications can process a queue simultaneously.

Checking the queue when the current depth is not increasing:

Even if the current depth of your queue is not increasing, it might still be useful to monitor the queue to check whether your application is processing messages correctly.

About this task

To gather performance data about the queue: Issue the following command periodically:

Procedure

Issue the following command periodically:
```
DISPLAY QSTATUS(Q1) TYPE(QUEUE) MSGAGE QTIME
```

In the output, if the value in MSGAGE increases over the period of time, and your application is designed to process all messages, this might indicate that some messages are not being processed at all.

Monitoring channels

Use this page to view tasks that help you to resolve a problem with a transmission queue and the channel that services that queue. Various channel monitoring options are available to determine the problem.

Frequently, the first sign of a problem with a queue that is being serviced is that the number of messages on the queue (CURDEPTH) increases. If you expect an increase at certain times of day or under certain workloads, an increasing number of messages might not indicate a problem. However, if you have no explanation for the increasing number of messages, you might want to investigate the cause.

You might have a problem with the channel that services a transmission queue. Various channel monitoring options are available to help you to determine the problem.

The following examples investigate problems with a transmission queue called QM2 and a channel called QM1.TO.QM2. This channel is used to send messages from queue manager, QM1, to queue manager, QM2. The channel definition at queue manager QM1 is either a sender or server channel, and the channel definition at queue manager, QM2, is either a receiver or requester channel.

Determining whether the channel is running:

If you have a problem with a transmission queue, check whether the channel is running.

About this task

Perform the following steps to check the status of the channel that is servicing the transmission queue:

Procedure

1. Issue the following command to find out which channel you expect to process the transmission queue QM2:
   ```
   DIS CHANNEL(*) WHERE(XMITQ EQ QM2)
   ```
   In this example, the output of this command shows that the channel servicing the transmission queue is QM1.TO.QM2
2. Issue the following command to determine the status of the channel, QM1.TO.QM2:
   ```
   DIS CHSTATUS(QM1.TO.QM2) ALL
   ```
3. Inspect the STATUS field of the output from the **CHSTATUS** command:
   - If the value of the STATUS field is `RUNNING`, check that the channel is moving messages
   - If the output from the command shows no status, or the value of the STATUS field is `STOPPED`, `RETRY`, `BINDING`, or `REQUESTING`, perform the appropriate step, as follows:
4. Optional: If the value of the STATUS field shows no status, the channel is inactive, so perform the following steps:
   a. If the channel should have been started automatically by a trigger, check that the messages on the transmission queue are available. If there are messages available on the transmission queue, check that the trigger settings on the transmission queue are correct.
   b. Issue the following command to start the channel again manually:
   ```
   START CHANNEL(QM1.TO.QM2)
   ```
5. Optional: If the value of the STATUS field is `STOPPED`, perform the following steps:
   a. Check the error logs to determine why the channel stopped. If the channel stopped owing to an error, correct the problem. Ensure also that the channel has values specified for the retry attributes: *SHORTRTY* and *LONGRTY*. In the event of transient failures such as network errors, the channel will then attempt to restart automatically.
   b. Issue the following command to start the channel again manually:

```
START CHANNEL(QM1.TO.QM2)
```

z/OS On IBM MQ for z/OS, you can detect when a user stops a channel by using command event messages.

6. Optional: If the value of the STATUS field is `RETRY`, perform the following steps:
   a. Check the error logs to identify the error, then correct the problem.
   b. Issue the following command to start the channel again manually:
      ```
      START CHANNEL(QM1.TO.QM2)
      ```
      or wait for the channel to connect successfully on its next retry.
7. Optional: If the value of the STATUS field is `BINDING` or `REQUESTING`, the channel has not yet successfully connected to the partner. Perform the following steps:
   a. Issue the following command, at both ends of the channel, to determine the substate of the channel:
      ```
      DIS CHSTATUS(QM1.TO.QM2) ALL
      ```
      Note:
      1) In some cases there might be a substate at one end of the channel only.
      2) Many substates are transitory, so issue the command a few times to detect whether a channel is stuck in a particular substate.
   b. Check Table 131 to determine what action to take:

Table 131. Substates seen with status binding or requesting

Initiating MCA substate [1]	Responding MCA substate [2]	Notes
NAMESERVER		The initiating MCA is waiting for a name server request to complete. Ensure that the correct host name has been specified in the channel attribute, CONNAME, and that your name servers are set up correctly.
SCYEXIT	SCYEXIT	The MCAs are currently *in conversation* through a security exit. For more information, see “Determining whether the channel can process messages fast enough” on page 1217.
	CHADEXIT	The channel autodefinition exit is currently executing. For more information, see “Determining whether the channel can process messages fast enough” on page 1217.
RCVEXIT SENDEXIT MSGEXIT MREXIT	RCVEXIT SENDEXIT MSGEXIT MREXIT	Exits are called at channel startup for MQXR_INIT. Review the processing in this part of your exit if this takes a long time. For more information, see “Determining whether the channel can process messages fast enough” on page 1217.
SERIALIZE	SERIALIZE	This substate only applies to channels with a disposition of SHARED.
NETCONNECT		This substate is shown if there is a delay in connecting due to incorrect network configuration.
SSLHANDSHAKE	SSLHANDSHAKE	An SSL handshake consists of a number of sends and receives. If network times are slow, or connection to lookup CRLs are slow, this affects the time taken to do the handshake. z/OS On IBM MQ for z/OS this substate can also be indicative of not having enough SSLTASKS.

Notes:

1) The initiating MCA is the end of the channel which started the conversation. This can be senders, cluster-senders, fully-qualified servers and requesters. In a server-requester pair, it is the end from which you started the channel.
2) The responding MCA is the end of the channel which responded to the request to start the conversation. This can be receivers, cluster-receivers, requesters (when the server or sender is started), servers (when the requester is started) and senders (in a requester-sender call-back pair of channels).

Checking that the channel is moving messages:

If you have a problem with a transmission queue, check that the channel is moving messages

Before you begin

Issue the command `DIS CHSTATUS(QM1.TO.QM2) ALL`. If the value of the STATUS field is `RUNNING`, the channel has successfully connected to the partner system.

Check that there are no uncommitted messages on the transmission queue, as described in "Checking that messages on the queue are available" on page 1211.

About this task

If there are messages available for the channel to get and send, perform the following checks:

Procedure

1. In the output from the display channel status command, `DIS CHSTATUS(QM1.TO.QM2) ALL`, look at the following fields:

   MSGS
   Number of messages sent or received (or, for server-connection channels, the number of MQI calls handled) during this session (since the channel was started).

   BUFSSENT
   Number of transmission buffers sent. This includes transmissions to send control information only.

   BYTSSENT
   Number of bytes sent during this session (since the channel was started). This includes control information sent by the message channel agent.

   LSTMSGDA
   Date when the last message was sent or MQI call was handled, see LSTMSGTI.

   LSTMSGTI
   Time when the last message was sent or MQI call was handled. For a sender or server, this is the time the last message (the last part of it if it was split) was sent. For a requester or receiver, it is the time the last message was put to its target queue. For a server-connection channel, it is the time when the last MQI call completed.

   CURMSGS
   For a sending channel, this is the number of messages that have been sent in the current batch. For a receiving channel, it is the number of messages that have been received in the current batch. The value is reset to zero, for both sending and receiving channels, when the batch is committed.
2. Determine whether the channel has sent any messages since it started. If any have been sent, determine when the last message was sent.
3. If the channel has started a batch that has not yet completed, as indicated by a non-zero value in CURMSGS, the channel might be waiting for the other end of the channel to acknowledge the batch.

Look at the SUBSTATE field in the output and refer to Table 132:

Table 132. Sender and receiver MCA substates

Sender SUBSTATE	Receiver SUBSTATE	Notes
MQGET	RECEIVE	Normal states of a channel at rest.
SEND	RECEIVE	SEND is usually a transitory state. If SEND is seen it indicates that the communication protocol buffers have filled. This can indicate a network problem.
RECEIVE		If the sender is seen in RECEIVE substate for any length of time, it is waiting on a response, either to a batch completion or a heartbeat. You might want to check why a batch takes a long time to complete.

Note: You might also want to determine whether the channel can process messages fast enough, especially if the channel has a substate associated with exit processing.

Checking why a batch takes a long time to complete:

Reasons why a batch can take a long time to complete include a slow network or a channel is using message retry processing.

About this task

When a sender channel has sent a batch of messages it waits for confirmation of that batch from the receiver, unless the channel is pipelined. The factors described in this task can affect how long the sender channel waits.

Procedure

- Check whether the network is slow. The NETTIME value is the amount of time, displayed in microseconds, taken to send an end of batch request to the remote end of the channel and receive a response minus the time to process the end of batch request. This value can be large for either of the following reasons:
  - The network is slow. A slow network can affect the time it takes to complete a batch. The measurements that result in the indicators for the NETTIME field are measured at the end of a batch. However, the first batch affected by a slowdown in the network is not indicated with a change in the NETTIME value because it is measured at the end of the batch.
  - Requests are queued at the remote end, for example a channel can be retrying a put, or a put request may be slow due to page set I/O. Once any queued requests have completed, the duration of the end of batch request is measured. So if you get a large NETTIME value, check for unusual processing at the remote end.
- Check whether the channel is using message retry. If the receiver channel fails to put a message to a target queue, it might use message retry processing, rather than put the message to a dead-letter immediately. Retry processing can cause the batch to slow down. In between MQPUT attempts, the channel will have STATUS(PAUSED), indicating that it is waiting for the message retry interval to pass.

Determining whether the channel can process messages fast enough:

If there messages are building up on the transmission queue, but you have found no processing problems, determine whether the channel can process messages fast enough.

Before you begin

Issue the following command repeatedly over a period of time to gather performance data about the channel:

```
DIS CHSTATUS(QM1.TO.QM2) ALL
```

About this task

Confirm that there are no uncommitted messages on the transmission queue, as described in “Checking that messages on the queue are available” on page 1211, then check the XQTIME field in the output from the display channel status command. When the values of the XQTIME indicators are consistently high, or increase over the measurement period, the indication is that the channel is not keeping pace with the putting applications.

Perform the following tests:

Procedure

1. Check whether exits are processing. If exits are used on the channel that is delivering these messages, they might add to the time spent processing messages. To identify if this is the case, do the following checks:
   a. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the EXITTIME field. If the time spent in exits is higher than expected, review the processing in your exits for any unnecessary loops or extra processing, especially in message, send, and receive exits. Such processing affects all messages moved across the channel.
   b. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the SUBSTATE field. If the channel has of one of the following substates for a significant time, review the processing in your exits:
      - SCYEXIT
      - RCVEXIT
      - SENDEXIT
      - MSGEXIT
      - MREXIT
2. Check whether the network is slow. If messages are not moving fast enough across a channel, it might be because the network is slow. To identify if this is the case, do the following checks:
   a. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the NETTIME field. These indicators are measured when the sending channel asks its partner for a response. This happens at the end of each batch and, when a channel is idle during heartbeating.
   b. If this indicator shows that round trips are taking longer than expected, use other network monitoring tools to investigate the performance of your network.
3. Check whether the channel is using compression. If the channel is using compression, this adds to the time spent processing messages. If the channel is using only one compression algorithm, do the following checks:
   a. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the COMPTIME field. These indicators show the time spent during compression or decompression.
   b. If the chosen compression is not reducing the amount of data to send by the expected amount, change the compression algorithm.
4. If the channel is using multiple compression algorithms, do the following checks:

a. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the COMPTIME, COMPHDR, and COMPMSG fields.
b. Change the compression algorithms specified on the channel definition, or consider writing a message exit to override the channel's choice of compression algorithm for particular messages if the rate of compression, or choice of algorithm, is not providing the required compression or performance.

Solving problems with cluster channels:

If you have a build up of messages on the SYSTEM.CLUSTER.TRANSMIT.QUEUE queue, the first step in diagnosing the problem is discovering which channel, or channels, are having a problem delivering messages.

About this task

To discover which channel, or channels, using the SYSTEM.CLUSTER.TRANSMIT.QUEUE are having a problem delivering messages. Perform the following checks:

Procedure

1. Issue the following command:
   ```
   DIS CHSTATUS(*) WHERE(XQMSGSA GT 1)
   ```
   Note: If you have a busy cluster that has many messages moving, consider issuing this command with a higher number to eliminate the channels that have only a few messages available to deliver.
2. Look through the output for the channel, or channels, that have large values in the field XQMSGSA. Determine why the channel is not moving messages, or is not moving them fast enough. Use the tasks outlined in "Monitoring channels" on page 1213 to diagnose the problems with the channels found to be causing the build up.

The Windows performance monitor

In IBM WebSphere MQ Version 7.0 and earlier versions, it was possible to monitor the performance of local queues on Windows systems by using the Windows performance monitor. As of IBM WebSphere MQ Version 7.1, this method of performance monitoring is no longer available.

You can monitor queues on all supported platforms by using methods described in "Real-time monitoring" on page 1206.

Monitoring clusters

Within a cluster you can monitor application messages, control messages, and logs. There are special monitoring ocnsiderations when the cluster load balances between two or more instances of a queue.

Monitoring application messages in the cluster

Typically, all cluster messages that leave the queue manager pass through the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, irrespective of which cluster sender channel is being used to transmit the message. Each channel is draining messages targeted for that channel in parallel with all other cluster sender channels. A growing build-up of messages on this queue can indicate a problem with one or more channels and must be investigated:

- The depth of the queue must be monitored appropriately for the cluster design.
- The following command returns all channels that have more than one message that is waiting on the transmit queue:
  ```
  DIS CHSTATUS(*) WHERE(XQMSGSA GT 1)
  ```

With all cluster messages on a single queue, it is not always easy to see which channel has problems when it begins to fill up. Using this command is an easy way to see which channel is responsible.

You can configure a cluster queue manager to have multiple transmission queues. If you change the queue manager attribute DEFCLXQ to CHANNEL, every cluster-sender channel is associated with a different cluster transmit queue. Alternatively you can configure separate transmission queues manually. To display all the cluster transmit queues that are associated with cluster-sender channels, run the command:

```
DISPLAY CLUSQMGR (qmgrName) XMITQ
```

Define cluster transmission queues so that they follow the pattern of having the fixed stem of the queue name on the left. You can then query the depth of all the cluster transmission queues returned by the **DISPLAY CLUSMGR** command, by using a generic queue name:

```
DISPLAY QUEUE (qname *) CURDEPTH
```

Monitoring control messages in the cluster

The SYSTEM.CLUSTER.COMMAND.QUEUE queue is used for processing all cluster control messages for a queue manager, either generated by the local queue manager or sent to this queue manager from other queue managers in the cluster. When a queue manager is correctly maintaining its cluster state, this queue tends toward zero. There are situations where the depth of messages on this queue can temporarily grow however:

- Having lots of messages on the queue indicates churn in the cluster state.
- When making significant changes, allow the queue to settle in between those changes. For example, when moving repositories, allow the queue to reach zero before moving the second repository.

While a backlog of messages exists on this queue, updates to the cluster state or cluster-related commands are not processed. If messages are not being removed from this queue for a long time, further investigation is required, initially through inspection of the queue manager error logs z/OS (or CHINIT logs on z/OS) which might explain the process that is causing this situation.

The SYSTEM.CLUSTER.REPOSITORY.QUEUE holds the cluster repository cache information as a number of messages. It is usual for messages to always exist on this queue, and more for larger clusters. Therefore, the depth of messages on this queue is not an issue for concern.

Monitoring logs

Problems that occur in the cluster might not show external symptoms to applications for many days (and even months) after the problem originally occurs due to the caching of information and the distributed nature of clustering. However, the original problem is often reported in the IBM MQ error logs z/OS (and CHINIT logs on z/OS). For this reason, it is vital to actively monitor these logs for any messages written that relate to clustering. These messages must be read and understood, with any action taken where necessary.

For example: A break in communications with a queue manager in a cluster can result in knowledge of certain cluster resources that are being deleted due to the way that clusters regularly revalidate the cluster resources by republishing the information. A warning of such an event potentially occurring is reported by the message AMQ9465 z/OS or CSQX465I on z/OS systems. This message indicates that the problem needs to be investigated.

Special considerations for load balancing

When the cluster load balances between two or more instances of a queue, consuming applications must be processing messages on each of the instances. If one or more of those consuming applications terminates or stops processing messages, it is possible that clustering might continue to send messages to those instances of the queue. In this situation, those messages are not processed until the applications are functioning correctly again. For this reason the monitoring of the applications is an important part of the solution and action must be taken to reroute messages in that situation. An example of a mechanism to automate such monitoring can be found in this sample: The Cluster Queue Monitoring sample program (AMQSCLM).

Related concepts:

"Tuning distributed publish/subscribe networks" on page 1263
Use the tuning tips in this section to help improve the performance of your IBM MQ distributed publish/subscribe clusters and hierarchies.

"Balancing producers and consumers in publish/subscribe networks" on page 1267
An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

Monitoring performance and resource usage

Use this topic to understand the facilities available to monitor the performance, and resource usage of your IBM MQ for z/OS subsystems.

Related information:

Configuring z/OS

Administering IBM MQ for z/OS

Introduction to monitoring

z/OS

Use this topic as an overview of the monitoring facilities available for IBM MQ for z/OS. For example, obtaining snapshots, using IBM MQ trace, online monitoring, and events.

This topic describes how to monitor the performance and resource usage of IBM MQ.
- It outlines some of the information that you can retrieve and briefly describes a general approach to investigating performance problems. z/OS (You can find information about dealing with performance problems in the Problem determination on z/OS .)
- It describes how you can collect statistics about the performance of IBM MQ by using SMF records.
- It describes how to gather accounting data to enable you to charge your customers for their use of your IBM MQ systems.
- It describes how to use IBM MQ events (alerts) to monitor your systems.

Here are some of the tools you might use to monitor IBM MQ; they are described in the sections that follow:
- Tools provided by IBM MQ:
  - Using DISPLAY commands
  - "Using CICS adapter statistics" on page 1222
  - "Using IBM MQ events" on page 1224
- z/OS service aids:
  - "Using System Management Facility" on page 1224
- Other IBM licensed programs:
  - Using the Resource Measurement Facility
  - Using Tivoli Decision Support for z/OS
  - Using the CICS monitoring facility

Information about interpreting the data gathered by the performance statistics trace is given in“Interpreting IBM MQ performance statistics” on page 1227.

Information about interpreting the data gathered by the accounting trace is given in“Interpreting IBM MQ accounting data” on page 1251.

Getting snapshots of IBM MQ using the DISPLAY commands:

IBM MQ provides the MQSC facility which can give a snapshot of the performance, and resource usage using the DISPLAY commands.

You can get an idea of the current state of IBM MQ by using the DISPLAY commands and, for the CICS adapter, the CICS adapter panels.

Using DISPLAY commands

You can use the IBM MQ MQSC DISPLAY or PCF Inquire commands to obtain information about the current state of IBM MQ. They provide information about the status of the command server, process definitions, queues, the queue manager, and its associated components. These commands are:

MQSC command	PCF command
DISPLAY ARCHIVE	Inquire Archive
DISPLAY AUTHINFO	Inquire Authentication Information Object
DISPLAY CFSTATUS	Inquire CF Structure Status
DISPLAY CFSTRUCT	Inquire CF Structure
DISPLAY CHANNEL	Inquire Channel
DISPLAY CHINIT	Inquire Channel Initiator
DISPLAY CHSTATUS	Inquire Channel Status
DISPLAY CMDSERV	
DISPLAY CLUSQMGR	Inquire Cluster Queue Manager
DISPLAY CONN	Inquire Connection
DISPLAY GROUP	Inquire Group
DISPLAY LOG	Inquire Log
DISPLAY PROCESS	Inquire Process
DISPLAY QMGR	Inquire Queue Manager
DISPLAY QSTATUS	Inquire Queue Status
DISPLAY QUEUE	Inquire Queue
DISPLAY SECURITY	Inquire Security
DISPLAY STGCLASS	Inquire Storage Class
DISPLAY SYSTEM	Inquire System
DISPLAY TRACE	
DISPLAY USAGE	Inquire Usage

For the detailed syntax of each command, see MQSC commands or PCF commands. All of the functions of these commands (except DISPLAY CMDSERV and DISPLAY TRACE) are also available through the operations and control panels.

These commands provide a snapshot of the system *only* at the moment the command was processed. If you want to examine trends in the system, you must start an IBM MQ trace and analyze the results over a period of time.

Using CICS adapter statistics:

If you are an authorized CICS user, you can use the CICS adapter control panels to display CICS adapter statistics dynamically.

These statistics provide a snapshot of information related to CICS thread usage and situations when all threads are busy. The display connection panel can be refreshed by pressing the Enter key. For more information, see "The CICS-IBM MQ Adapter" section in the CICS Transaction Server for z/OS Version 4.1 product documentation at: CICS Transaction Server for z/OS Version 4.1, The CICS-IBM MQ adapter.

Using IBM MQ trace:

You can record performance statistics and accounting data for IBM MQ by using the IBM MQ trace facility. Use this topic to understand how to control IBM MQ trace.

The data generated by IBM MQ is sent to:

- The System Management Facility (SMF), specifically as SMF record type 115, subtypes 1 and 2 for the performance statistics trace
- The SMF, specifically as SMF record type 116, subtypes zero, 1, and 2 for the accounting trace.

If you prefer, the data generated by the IBM MQ accounting trace can also be sent to the generalized trace facility (GTF).

Starting IBM MQ trace

You can start the IBM MQ trace facility at any time by issuing the IBM MQ START TRACE command.

Accounting data can be lost if the accounting trace is started or stopped while applications are running. To collect accounting data successfully, the following conditions must apply:

- The accounting trace must be active when an application starts, and it must still be active when the application finishes.
- If the accounting trace is stopped, any accounting data collection that was active stops.

You can also start collecting some trace information automatically if you specify YES on the SMFSTAT (SMF STATISTICS) and SMFACCT (SMF ACCOUNTING) parameters of the CSQ6SYSP macro.
z/OS These parameters are described in Using CSQ6SYSP.

You cannot use this method to start collecting class 3 accounting information (thread-level and queue-level accounting). You must use the START TRACE command to collect such information. However, you can include the command in your CSQINP2 input data set so that the trace is started automatically when you start your queue manager.

Before starting an IBM MQ trace, read "Using System Management Facility" on page 1224.

Controlling IBM MQ trace

To control the IBM MQ trace data collection at start-up, specify values for the parameters in the CSQ6SYSP macro when you customize IBM MQ. z/OS See Using CSQ6SYSP for details.

You can control IBM MQ tracing when the queue manager is running with these commands:

- START TRACE
- ALTER TRACE
- STOP TRACE

You can choose the destination to which trace data is sent. Possible destinations are:

SMF System Management Facility

GTF Generalized Trace Facility (accounting trace only)

SRV Serviceability routine for diagnostic use by IBM service personnel

For daily monitoring, information is sent to SMF (the default destination). SMF data sets typically contain information from other systems; this information is not available for reporting until the SMF data set is dumped.

You can also send accounting trace information to the GTF. This information has an event identifier of 5EE. z/OS The The MQI call and user parameter, and z/OS generalized trace facility (GTF) describes how to deal with IBM MQ trace information sent to the GTF.

For information about IBM MQ commands, see MQSC commands.

Effect of trace on IBM MQ performance

Using the IBM MQ trace facility can have a significant effect on IBM MQ and transaction performance. For example, if you start a global trace for class 1 or for all classes, it is likely to increase processor usage and transaction response times by approximately 50%. However, if you start a global trace for classes 2 - 4 alone, the increase in processor usage and transaction response times is likely to be less than 1% additional processor cost to the cost of IBM MQ calls. The same applies for a statistics or accounting trace.

Using IBM MQ online monitoring:

You can collect monitoring data for queues and channels (including automatically defined cluster-server channels) by setting the MONQ, MONCHL, and MONACLS attributes.

Table 133 summarizes the commands to set these attributes at different levels and to display the monitoring information.

Table 133. Setting and displaying attributes to control online monitoring

Attribute	Applicable at this level	Set using command	Display monitoring information using command
MONQ	Queue	DEFINE QLOCAL DEFINE QMODEL ALTER QLOCAL ALTER QMODEL	DISPLAY QSTATUS
	Queue manager	ALTER QMGR	
MONCHL	Channel	DEFINE CHANNEL ALTER CHANNEL	DISPLAY CHSTATUS
	Queue manager	ALTER QMGR	
MONACLS	Queue manager	ALTER QMGR	

For full details of these commands, see MQSC commands. For more information about online monitoring, see “Monitoring your IBM MQ network” on page 929.

Using IBM MQ events:

IBM MQ instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. You can monitor the operation of all your queue managers by incorporating these events into your own system management application.

IBM MQ instrumentation events fall into the following categories:

Queue manager events
These events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

Performance events
These events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached, or the queue was not serviced within a predefined time limit.

Channel events
These events are reported by channels as a result of conditions detected during their operation. For example, a channel instance is stopped.

Configuration events
These events are notifications that an object has been created, changed, or deleted.

When an event occurs, the queue manager puts an *event message* on the appropriate *event queue*, if defined. The event message contains information about the event that can be retrieved by a suitable IBM MQ application.

IBM MQ events can be enabled using the IBM MQ commands or the operations and control panels.

See "Event types" on page 932 for information about the IBM MQ events that generate messages, and for information about the format of these messages. See Event message reference for information about enabling the events.

Using System Management Facility: z/OS

You can use SMF to collect statistics and accounting information. To use SMF, certain parameters must be set in z/OS and in IBM MQ.

System management facility (SMF) is a z/OS service aid used to collect information from various z/OS subsystems. This information is dumped and reported periodically, for example, hourly. You can use SMF with the IBM MQ trace facility to collect data from IBM MQ. In this way you can monitor *trends*, for example, in system utilization and performance, and collect accounting information about each user ID using IBM MQ.

To record performance statistics (record type 115) to SMF specify the following in the SMFPRMxx member of SYS1.PARMLIB or with the SETSMF z/OS operator command.

```
SYS(TYPE(115))
```

To record accounting information (record type 116) to SMF specify the following in the SMFPRMxx member of SYS1.PARMLIB or with the SETSMF z/OS operator command.

```
SYS(TYPE(116))
```

You can turn on or off the recording of accounting information at the queue or queue manager level using the ACCTQ parameter of the DEFINE QLOCAL, DEFINE QMODEL, ALTER QLOCAL, ALTER QMODEL, or ALTER QMGR commands. See MQSC commands for details of these commands.

To use the z/OS command SETSMF, either PROMPT(ALL) or PROMPT(LIST) must be specified in the SMFPRM *xx* member. See the z/OS MVS Initialization and Tuning Reference and the z/OS MVS System Commands manuals for more information.

You can start collecting some trace information automatically if you specify YES on the SMFSTAT (SMF STATISTICS) and SMFACCT (SMF ACCOUNTING) parameters of the CSQ6SYSP macro; this is described in Using CSQ6SYSP.

Specifying YES on the SMFSTAT and SMFACCT parameters enables you to collect trace information as a queue manager starts.

You can also start collection of the data on a queue manager by specifying START TRACE(A) or START TRACE(S).

You can specify the interval at which IBM MQ collects statistics and accounting data in one of two ways:

- You can specify a value for STATIME in your system parameters z/OS (described in Using CSQ6SYSP).
- You can specify zero for STATIME and use the SMF global accounting interval (described in the z/OS MVS Initialization and Tuning Reference).

SMF must be running before you can send data to it. For more information about SMF, see the MVS System Management Facilities (SMF) manual.

For the statistics and accounting data to be reset, at least one MQI call must be issued during the accounting interval.

Allocating additional SMF buffers

When you start a trace, you must ensure that you allocate adequate SMF buffers. Specify SMF buffering on the VSAM BUFSP parameter of the access method services DEFINE CLUSTER statement. Specify CISZ(4096) and BUFSP(81920) on the DEFINE CLUSTER statement for each SMF VSAM data set.

If an SMF buffer shortage occurs, SMF rejects any trace records sent to it. IBM MQ sends a CSQW133I message to the z/OS console when this occurs. IBM MQ treats the error as temporary and remains active even though SMF data can be lost. When the shortage has been alleviated and trace recording has resumed, IBM MQ sends a CSQW123I message to the z/OS console.

Reporting data in SMF

You can use the SMF program IFASMFDP to dump SMF records to a sequential data set so that they can be processed.

There are several ways to report on this data, for example:

- Write an application program to read and report information from the SMF data set. You can then tailor the report to fit your exact needs.
- Use Performance Reporter to process the records. For more information, see “Using other products with IBM MQ” on page 1226.

Using other products with IBM MQ:

You can use other products to help you to improve the presentation of, or to augment statistics related to, performance and accounting. For example, Resource Measurement Facility, Tivoli Decision Support, and CICS monitoring.

Using the Resource Measurement Facility

Resource Measurement Facility (RMF) is an IBM licensed program (program number 5685-029) that provides system-wide information about processor utilization, I/O activity, storage, and paging. You can use RMF to monitor the utilization of physical resources across the whole system dynamically. For more information, see the *MVS Resource Measurement Facility User's Guide*.

Using Tivoli Decision Support for z/OS

You can use Tivoli Decision Support for z/OS to interpret RMF and SMF records.

Tivoli Decision Support for z/OS is an IBM licensed program (program number 5698-B06) that enables you to manage the performance of your system by collecting performance data in a Db2 database and presenting the data in various formats for use in systems management. Tivoli Decision Support for can generate graphic and tabular reports using systems management data it stores in its Db2 database. It includes an administration dialog, a reporting dialog, and a log collector, all of which interact with a standard Db2 database.

This is described in the *Tivoli Decision Support Administrator's Guide*.

Using the CICS monitoring facility

The CICS monitoring facility provides performance information about each CICS transaction running. It can be used to investigate the resources used and the time spent processing transactions. For background information, see the *CICS Performance Guide* and the *CICS Customization Guide*.

Investigating performance problems:

Performance problems can arise from various factors. For example, incorrect resource allocation, poor application design, and I/O restraints. Use this topic to investigate some of the possible causes of performance problems.

Performance can be adversely affected by:
- Buffer pools that are an incorrect size
- Lack of real storage
- I/O contention for page sets or logs
- Log buffer thresholds that are set incorrectly
- Incorrect setting of the number of log buffers
- Large messages
- Units of recovery that last a long time, incorporating many messages for each sync point
- Messages that remain on a queue for a long time
- RACF auditing
- Unnecessary security checks
- Inefficient program design

When you analyze performance data, always start by looking at the overall system before you decide that you have a specific IBM MQ problem. Remember that almost all symptoms of reduced performance are magnified when there is contention. For example, if there is contention for DASD, transaction response times can increase. Also, the more transactions there are in the system, the greater the processor usage and greater the demand for both virtual and real storage.

In such situations, the system shows heavy use of *all* its resources. However, the system is actually experiencing normal system stress, and this stress might be hiding the cause of a performance reduction. To find the cause of such a loss of performance, you must consider all items that might be affecting your active tasks.

Investigating the overall system

Within IBM MQ, the performance problem is either increased response time or an unexpected and unexplained heavy use of resources. First check factors such as total processor usage, DASD activity, and paging. An IBM tool for checking total processor usage is resource management facility (RMF). In general, you must look at the system in some detail to see why tasks are progressing slowly, or why a specific resource is being heavily used.

Start by looking at general task activity, then focus on particular activities, such as specific tasks or a specific time interval.

Another possibility is that the system has limited real storage; therefore, because of paging interrupts, the tasks progress more slowly than expected.

Investigating individual tasks

You can use the accounting trace to gather information about IBM MQ tasks. These trace records tell you a great deal about the activity that the task has performed, and about how much time the task spent suspended, waiting for latches. The trace record also includes information about how much Db2 and coupling facility activity were performed by the task.

Interpreting IBM MQ accounting data is described in “Interpreting IBM MQ accounting data” on page 1251.

Long running units of work can be identified by the presence of message CSQR026I in the job log. This message indicates that a task has existed for more than three queue manager checkpoints and its log records have been shunted. z/OS For a description of log record shunting, see The log files.

Interpreting IBM MQ performance statistics

Use this topic as an index to the different SMF records created by IBM MQ for z/OS.

IBM MQ performance statistics are written as SMF type 115 records. Statistics records are produced periodically at a time interval specified by the STATIME parameter of the CSQ6SYSP system parameter module, or at the SMF global accounting interval if you specify zero for STATIME. The information provided in the SMF records comes from the following components of IBM MQ:

Buffer manager	Manages the buffer pools in virtual storage and the writing of pages to page sets as the buffer pools become full. Also manages the reading of pages from page sets.
Coupling facility manager	Manages the interface with the coupling facility.
Data manager	Manages the links between messages and queues. It calls the buffer manager to process the pages with messages on them.
Db2 manager	Manages the interface with the Db2 database that is used as the shared repository.
Lock manager	Manages locks for IBM MQ for z/OS.

Log manager	Manages the writing of log records, which are essential for maintaining the integrity of the system if there is a back out request, or for recovery, if there is a system or media failure.
Message manager	Processes all IBM MQ API requests.
Storage manager	Manages storage for IBM MQ for z/OS, for example, storage pool allocation, expansion, and deallocation.
Topic manager	Manages the Topic and Subscription information for IBM MQ for z/OS.
Coupling facility SMDS manager	Manages the shared message data sets (SMDS) for large messages stored in the coupling facility.

IBM MQ statistics are written to SMF as SMF type 115 records. The following subtypes can be present:

1 System information, for example, related to the logs and storage.

2 Information about number of messages, buffer, and paging information. Queue-sharing group information related to the coupling facility and Db2.

3 More detailed information about storage usage in the MSTR address space.

215 Bufferpool information when the queue manager is running in `OPMODE=(NEWFUNC,800)` mode.

231 System information for the CHIN address space.

Note that:

- Subtype 1, 2, and 215 are created with statistics trace class(1)
- Subtype 3 with statistics trace classes 2 and 3
- Subtype 231 with statistics trace class(4)

The subtype is specified in the SM115STF field (shown in Table 134 on page 1229).

Layout of an SMF type 115 record:

You can use this section as a reference for the format of an SMF type 115 record.

The standard layout for SMF records involves three parts:

SMF header
Provides format, identification, and time and date information about the record itself.

Self-defining section
Defines the location and size of the individual data records within the SMF record.

Data records
The actual data from IBM MQ that you want to analyze.

For more information about SMF record formats, see the *MVS System Management Facilities (SMF)* manual.

Related reference:

"The SMF header"
Use this topic as a reference for the format of the SMF header.

"Self-defining sections" on page 1230
Use this topic as a reference for format of the self-defining sections of the SMF record.

"Examples of SMF statistics records" on page 1231
Use this topic to understand some example SMF records.

The SMF header:

Use this topic as a reference for the format of the SMF header.

Table 134 shows the format of SMF record header (SM115).

Table 134. SMF record 115 header description

Offset: Dec	Offset: Hex	Type	Len	Name	Description	Example
0	0	Structure	28	SM115	SMF record header.	
0	0	Integer	2	SM115LEN	SMF record length.	14A0
2	2		2		Reserved.	
4	4	Integer	1	SM115FLG	System indicator.	5E
5	5	Integer	1	SM115RTY	Record type. The SMF record type, for IBM MQ statistics records this is always 115 (X'73').	73
6	6	Integer	4	SM115TME	Time when SMF moved record.	00355575
10	A	Integer	4	SM115DTE	Date when SMF moved record.	0100223F
14	E	Character	4	SM115SID	z/OS subsystem ID. Defines the z/OS subsystem on which the records were collected.	D4E5F4F1 (MV41)
18	12	Character	4	SM115SSI	IBM MQ subsystem ID.	D4D8F0F7 (MQ07)
22	16	Integer	2	SM115STF	Record subtype.	0002
24	18	Character	3	SM115REL	IBM MQ version.	F6F0F0 (600)
27	1B		1		Reserved	
28	1C	Character	0	SM115END	End of SMF header and start of self-defining section.	

Self-defining sections:

Use this topic as a reference for format of the self-defining sections of the SMF record.

A self-defining section of a type 115 SMF record tells you where to find a statistics record, how long it is, and how many times that type of record is repeated (with different values). The self-defining sections follow the header, at fixed offsets from the start of the SMF record. Each statistics record can be identified by an eye-catcher string.

The following types of self-defining section are available to users for type 115 records. Each self-defining section points to statistics data related to one of the IBM MQ components. Table 135 summarizes the sources of the statistics, the eye-catcher strings, and the offsets of the self-defining sections from the start of the SMF record header.

Table 135. Offsets to self-defining sections

Source of statistics	**Record subtype (SM115STF)**	**Offset of self-defining section**		**Eye-catcher of data**
		Dec	**Hex**	
Storage manager	1	100	X'64'	QSST
Log manager	1	116	X'74'	QJST
Message manager	2	36	X'24'	QMST
Data manager	2	44	X'2C'	QIST
Buffer manager - one for each buffer pool, see note below	2	52	X'34'	QPST
Lock manager	2	60	X'3C'	QLST
Db2 manager	2	68	X'44'	Q5ST
Coupling Facility manager	2	76	X'4C'	QEST
Topic manager	2	84	X'54'	QTST
SMDS usage	2	92	X'5C'	QESD
Buffer manager - one for each buffer pool, see note below	215	36	X'24'	QPST
Channel initiator	231			QWSX

Notes:

1. For the Buffer Manager entry in subtype 2 the self-defining section will always show zeros indicating that there are no buffer manager records.
2. Other self-defining sections refer to data for IBM use only.

Buffer manager statistics go into different subtypes depending on the settings for OPMODE in the system parameters. If `OPMODE(NEWFUNC,800)` is in effect, buffer manager statistics will go into subtype 215, and the self defining section for buffer manager statistics in subtype 2 will be all zeros. If `OPMODE(COMPAT,800)`, or equivalent, is in effect, buffer manager statistics will go into subtype 2 and subtype 215 statistics will not be generated.

Each self-defining section is two fullwords long and has this format:

```
ssssssssllllnnnn
```

where:

ssssssss	Fullword containing the offset from the start of the SMF record.
llll	Halfword giving the length of this data record.
nnnn	Halfword giving the number of data records in this SMF record.

For more information see, "Examples of SMF statistics records."

Note: Always use offsets in the self-defining sections to locate the statistics records.

Examples of SMF statistics records:

Use this topic to understand some example SMF records.

Figure 105 shows an example of part of the SMF record for subtype 1. Subtype 1 includes the storage manager and log manager statistics records. The SMF record header is shown underlined.

The self-defining section at offset X'64' refers to storage manager statistics and the self-defining section at offset X'74' refers to log manager statistics, both shown in **bold**.

The storage manager statistics record is located at offset X'0000011C' from the start of the header and is X'48' bytes long. There is one set of storage manager statistics, identified by the eye-catcher string QSST. The start of this statistics record is also shown in the example.

The log manager statistics record is located at offset X'00000164' from the start of the header and is X'78' bytes long. There is one set of log manager statistics, identified by the eye-catcher string QJST.

```
000000  02000000 5E730035 55750100 223FD4E5  *....;.........MV*
000010  F4F1D4D8 F0F70001 F6F0F000 000001DC  *41MQ07..600.....*
000020  00240001 00000000 00000000 00000000  *................*
000030  00000000 00000000 00000000 0000007C  *...............@*
000040  00400001 000000BC 00600001 00000000  *. .......-......*
000050  00000000 00000000 00000000 00000000  *................*
000060  00000000 0000011C 00480001 00000000  *................*
000070  00000000 00000164 00780001 00000000  *................*
000080  00000000 00000000 00000000 00000000  *................*
.
.
000110  00000000 00000000 00000000 003C0048  *................*
000120  D8E2E2E3 0000004F 00000003 00000002  *QSST...|........*
```

Figure 105. SMF record 115, subtype 1

Figure 106 on page 1232 shows an example of part of the SMF record for subtype 2. Subtype 2 includes the statistics records for the message, data, lock, coupling facility, topic, and Db2 managers. The SMF record header is shown underlined; the self-defining sections are shown alternately **bold** and *italic*.

- The self-defining section at offset X'24' refers to message manager statistics. The message manager statistics record is located at offset X'00000064' from the start of the header and is X'48' bytes long. There is one set of these statistics, identified by the eye-catcher string QMST.
- The self-defining section at offset X'2C' refers to data manager statistics. The data manager statistics record is located at offset X'000000AC' from the start of the header and is X'50' bytes long. There is one set of these statistics, identified by the eye-catcher string QIST.

- The self-defining section at offset X'34' refers to buffer manager statistics. As this SMF record was taken from a queue manager that has OPMODE(NEWFUNC,800) set in its system parameters, the buffer manager self-defining section is set to zeros to indicate that there are no buffer manager statistics. Instead, these statistics are in SMF 115 subtype 215 records.
- The self-defining section at offset X'3C' refers to lock manager statistics. The lock manager statistics record is located at offset X'000000FC' from the start of the header and is X'20' bytes long. There is one set of these statistics, identified by the eye-catcher string QLST.
- The self-defining section at offset X'44' refers to Db2 manager statistics. The Db2 manager statistics record is located at offset X'0000011C' from the start of the header and is X'2A0' bytes long. There is one set of these statistics, identified by the eye-catcher string Q5ST.
- The self-defining section at offset X'4C' refers to coupling facility manager statistics. The coupling facility manager statistics record is located at offset X'000003BC' from the start of the header and is X'1008' bytes long. There is one set of these statistics, identified by the eye-catcher string QEST.
- The self-defining section at offset X'54' refers to topic manager statistics. The topic manager statistics record is located at offset X'000013C4' from the start of the header and is X'64' bytes long. There is one set of these statistics, identified by the eye-catcher string QTST.
- The self-defining section at offset X'5C' is for SMDS statistics. This self defining section is set to zeros indicating that SMDS is not being used.

```
000000 09F40000 5E730033 4DBE0113 142FD4E5 *.4..;...(.....MV*
000010 F4F1D4D8 F2F10002 F8F0F000 00001428 *41MQ21..800.....*
000020 00240001 00000064 00480001 000000AC *................*
000030 00500001 00000000 00000000 000000FC *................*
000040 00200001 0000011C 02A00001 000003BC *................*
000050 10080001 000013C4 00640001 00000000 *.......D........*
000060 00000000 D40F0048 D8D4E2E3 00000000 *....M...QMST....*
000080 00000000 00000000 00000000 00000000 *................*
000090 00000000 00000000 00000000 00000000 *................*
0000A0 00000000 00000000 00000000 C90F0050 *............I..&*
0000B0 D8C9E2E3 00000000 00000000 00000000 *QIST............*
0000C0 00000000 00000000 00000000 00000000 *................*
0000D0 00000000 00000000 00000000 00000000 *................*
0000E0 00000000 00000000 00000000 00000000 *................*
0000F0 00000000 00000000 00000000 D30F0020 *............L...*
000100 D8D3E2E3 00000000 00000000 00000000 *QLST............*
000110 00000000 00000000 00000000 F50F02A0 *............5...*
000120 D8F5E2E3 00000008 00000000 00000000 *Q5ST............*
.
.
```

Figure 106. SMF record 115, subtype 2

Processing type 115 SMF records:

Use this topic as a reference for processing type 115 SMF records.

You must process any data you collect from SMF to extract useful information. When you process the data, verify that the records are from IBM MQ and that they are the records you are expecting.

Validate the values of the following fields:
- SM115RTY, the SMF record number, must be X'73' (115)
- SM115STF, the record subtype, must be 0001, 0002, 0215, or 0231

Reading from the active SMF data sets is not supported. You must use the SMF program IFASMFDP to dump SMF record to a sequential data set so that they can be processed. For more information see "Using System Management Facility" on page 1224.

There is a C sample program called CSQ4SMFD which prints the contents of SMF type 115 and 116 records. The program is provided as source in thlqual.SCSQC37S and in executable format in thlqual.SCSQLOAD. Sample JCL is provided in thlqual.SCSQPROC(CSQ4SMFJ).

Storage manager data records:

Use this topic as a reference for storage manager data records.

The format of the storage manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQSST).

The data contains information about the number of fixed and variable storage pools that the queue manager has allocated, expanded, contracted, and deleted during the statistics interval, plus the number of GETMAIN, FREEMAIN, and STORAGE requests to z/OS, including a count of those requests that were unsuccessful. Additional information includes a count of the number of times the short-on-storage condition was detected and a count of the number of abends that occurred as a result of that condition.

Log manager data records:

Use this topic as a reference for format of log manager data records.

The format of the log manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQJST).

In the statistics, these counts are important:
1. The total number of log write requests:

   $N_{logwrite}$ = QJSTWRW + QJSTWRNW + QJSTWRF
2. The total number of log read requests:

   $N_{logread}$ = QJSTRBUF + QJSTRACT + QJSTRARH

The problem symptoms that can be examined using log manager statistics are described in the following table.

Symptom 1
QJSTWTB is nonzero.

Reason
Tasks are being suspended while the in-storage buffer is being written to the active log.

There might be problems writing to the active log.

The OUTBUFF parameter within CSQ6LOGP is too small.

Action
Investigate the problems writing to the active log.

Increase the value of the OUTBUFF parameter within CSQ6LOGP.

Symptom 2
The ratio: $\mathtt{QJSTWTL}/\mathtt{N}_{\mathtt{logread}}$ is greater than 1%.

Reason
Log reads were initiated that had to read from an archive log, but IBM MQ could not allocate a data set because MAXRTU data sets were already allocated.

Action
Increase MAXRTU.

Symptom 3
The ratio: $\mathtt{QJSTRARH}/\mathtt{N}_{\mathtt{logread}}$ is larger than normal.

Reason
Most log read requests should come from the output buffer or the active log. To satisfy requests for back out, unit-of-recovery records are read from the in-storage buffer, the active log, and the archived logs.

A long-running unit of recovery, extending over a period of many minutes, might have log records spread across many different logs. This degrades performance because extra work has to be done to recover the log records.

Action
Change the application to reduce the length of a unit of recovery. Also, consider increasing the size of the active log to reduce the possibility of a single unit of recovery being spread out over more than one log.

Other pointers
The ratio $\mathtt{N}_{\mathtt{logread}}/\mathtt{N}_{\mathtt{logwrite}}$ gives an indication of how much work has to be backed out.

Symptom 4
QJSTLLCP is more than 10 an hour.

Reason
On a busy system, you would expect to see typically 10 checkpoints an hour. If the QJSTLLCP value is larger than this, it indicates a problem in the setup of the queue manager.

The most likely reason for this is that the LOGLOAD parameter in CSQ6SYSP is too small. The other event that causes a checkpoint is when an active log fills up and switches to the next active log data set. If your logs are too small, this can cause frequent checkpoints. The QJSTLLCP counter is not incremented for log switch induced checkpoints; you must look in the JES logs for the queue managers to determine if the rate log files are switched.

Action
Increase the LOGLOAD parameter, or increase the size of your log data sets as required.

Symptom 5

QJSTCmpFail > 0 or QJSTCmpComp not much less than QJSTCmpUncmp

Reason

The queue manager is unable to significantly compress log records.

QJSTCmpFail is the number of times the queue manager was unable to achieve any reduction in record length. You should compare the number to QJSTCmpReq (number of compression requests) to see if the number of failures is significant.

QJSTCmpComp is the total of compressed bytes written to the log and QJSTCmpUncmp is the total bytes before compression. Neither total contains bytes written for log records that were not eligible for compression. If the numbers are similar then compression has achieved little benefit.

Action

Turn off log compression. Issue the SET LOG COMPLOG(NONE) command. See the SET LOG command for details.

Note: In the first set of statistics produced after system startup, there might be significant log activity due to the resolution of in-flight units of recovery.

Message manager data records:

Use this topic as a reference for message manager data records.

The format of the message manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQMST).

The data gives you counts of different IBM MQ API requests.

Data manager data records:

Use this topic as a reference for the format of the Data Manager data records.

The format of the data manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQIST).

The data gives you counts of different object requests.

Buffer manager data records:

Use this topic as a reference for the format of buffer manager data records.

The format of the buffer manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQPST).

Note: Buffer manager statistics records will only be created for buffer pools that are defined. If a buffer pool is defined but not used then no values will be set and its buffer manager statistics record will not contain any data.

For information about efficiently managing your buffer pools, see "Managing your buffer pools" on page 1237.

When interpreting the statistics, you are recommended to consider the following factors because the values of these fields can be used to improve the performance of your system:

1. If QPSTSOS, QPSTSTLA, or QPSTDMC is greater than zero, you should either increase the size of the buffer pool or reallocate the page sets to different buffer pools.

- QPSTSOS is the number of times that there were no buffers available for page get requests. If QPSTSOS ever becomes nonzero, it shows that IBM MQ is under severe stress. The buffer pool size should be increased. If increasing the buffer pool size does not make the value of QPSTSOS zero, there might be I/O contention on the DASD page sets.
- QPSTDMC is the number of updates that were performed synchronously because there was either more than 95% of the pages in the buffer pool waiting for write I/O, or there was less than 5% of the buffer pool available for read requests. If this number is not zero, the buffer pool might be too small and should be enlarged. If increasing the buffer pool size does not reduce QPSTDMC to zero, there might be I/O contention on the DASD page sets.
- QPSTIMW is a count of the number of times pages were written out synchronously. If QPSTDMC is zero, QPSTIMW is the number of times pages were found on the queue waiting for write I/O that had been there for at least two checkpoints.

2. For buffer pool zero and buffer pools that contain short-lived messages:
   - QPSTDWT should be zero, and the percentage QPSTCBSL/QPSTNBUF should be greater than 15%.

     QPSTDWT is the number of times the asynchronous write processor was started because there was either more than 85% of the pages in the buffer pool waiting for write I/O, or there was less than 15% of the buffer pool available for read requests. Increasing the buffer pool size should reduce this value. If it does not, the pattern of access is one of long delays between puts and gets.
   - QPSTTPW might be greater than zero due to checkpointing activity.
   - QPSTRIO should be zero unless messages are being read from a page set after the queue manager is restarted.

     The ratio of QPSTRIO to QPSTGETP shows the efficiency of page retrieval within the buffer pool. Increasing the buffer pool size should decrease this ratio and, therefore, increase the page retrieval efficiency. If this does not happen, it indicates that pages are not being frequently reaccessed. This implies a transaction pattern where there is a long delay between messages being put and then later retrieved.

     The ratio of QPSTGETN to QPSTGETP indicates the number of times an empty page, as opposed to a non-empty page, has been requested. This ratio is more an indication of transaction pattern, than a value that can be used to tune the system.
   - If QPSTSTL has a value greater than zero, this indicates that pages that have not been used before are now being used. This might be caused by an increased message rate, messages not being processed as fast as they were previously (leading to a buildup of messages), or larger messages being used.

     QPSTSTL is a count of the number of times a page access request did not find the page already in the buffer pool. Again, the lower the ratio of QPSTSTL to (QPSTGETP + QPSTGETN) is, the higher the page retrieval efficiency. Increasing the buffer pool size should decrease this ratio but, if it does not, it is an indication that there are long delays between puts and gets.
   - You are recommended to have sufficient buffers to handle your peak message rate.
3. For buffer pools with long-lived messages, where there are more messages than can fit into the buffer pool:
   - (QPSTRIO+QPSTWIO)/Statistics interval is the I/O rate to page sets. If this value is high, you should consider using multiple page sets on different volumes to allow I/O to be carried out in parallel.

     The higher the ratio of QPSTSTW to QPSTWIO, the better the efficiency of the asynchronous write processor. You can increase this ratio, and therefore the efficiency of the asynchronous write processor, by increasing the buffer pool size.
   - Over the period of time that the messages are processed (for example, if messages are written to a queue during the day and processed overnight) the number of read I/Os (QPSTRIO) should be approximately the total number of pages written (QPSTTPW). This shows that one page is read for every page written.

If QPSTRIO is much larger than QPSTTPW, this shows that pages are being read in multiple times. This might be a result of the application using **MQGET** by `MsgId` or `CorrelId` when the queue is not indexed, or browsing messages on the queue using get next.

The following actions might relieve this problem:

a. Increase the size of the buffer pool so that there are enough pages to hold the queue, in addition to any changed pages.
b. Use the INDXTYPE queue attribute, which allows a queue to be indexed by `MsgId` or `CorrelId` and eliminates the need for a sequential scan of the queue.
c. Change the design of the application to eliminate the use of **MQGET** with `MsgId` or `CorrelId`, or the get next with browse option.

   Note: Applications using long-lived messages typically process the first available message and do not use **MQGET** with `MsgId` or `CorrelId`, and they might browse only the first available message.
d. Move page sets to a different buffer pool to reduce contention between messages from different applications.

Managing your buffer pools:

To manage your buffer pools efficiently, you must consider the factors that affect the buffer pool I/O operations and also the statistics associated with the buffer pools.

The following factors affect buffer pool I/O operations.

- If a page containing the required data is not found in the buffer pool, it is read in synchronously to an available buffer from its DASD page set.
- Whenever a page is updated, it is put on an internal queue of pages to be (potentially) written out to DASD. This means that the buffer used by that page is unavailable for use by any other page until the buffer has been written to DASD.
- If the number of pages queued to be written to DASD exceeds 85% of the total number of buffers in the pool, an asynchronous write processor is started to put the buffers to DASD.

  Similarly, if the number of buffers available for page get requests become less than 15% of the total number of buffers in the pool, the asynchronous write processor is started to perform the write I/O operations.

  The write processor stops when the number of pages queued to be written to DASD has fallen to 75% of the total number of buffers in the pool.
- If the number of pages queued for writing to DASD exceeds 95% of the total number of buffers in the pool, all updates result in a synchronous write of the page to DASD.

  Similarly, if the number of buffers available for page get requests becomes less than 5% of the total number of buffers in the pool, all updates result in a synchronous write of the page to DASD.
- If the number of buffers available for page get requests ever reaches zero, a transaction that encounters this condition is suspended until the asynchronous write processor has finished.
- If a page is frequently updated, the page spends most of its time on the queue of pages waiting to be written to DASD. Because this queue is in least recently used order, it is possible that a frequently updated page placed on this least recently used queue is never written out to DASD. For this reason, at the time of update, if the page is found to have been waiting on the write operation to DASD queue for at least two checkpoints, it is synchronously written to DASD. Updating occurs at checkpoint time and is suspended until the asynchronous write processor has finished.

  The aim of this algorithm is to maximize the time pages spend in buffer pool memory while allowing the system to function if the system load puts the buffer pool usage under stress.

Lock manager data records:

Use this topic as a reference to the format of the lock manager data records.

The format of the lock manager statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQLST).

The records contain data about the following information:
- The number of lock get requests and lock release requests.
- The number of times a lock get request determined that the requested lock was already held.

Db2 manager data records:

Use this topic as a reference to the format of the Db2 manager data records.

The format of the Db2 manager statistics record is described in the following table and in assembler macro thlqual.SCSQMACS(CSQDQ5ST) and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example q5st, q5stid.

If the queue manager was not started as a member of a queue-sharing group, no data is recorded in this record.

Table 136. Db2 statistics record (Q5ST)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
0	0	Structure	668	Q5ST	Db2 manager statistics
0	0	Bitstring	2	Q5STID	Control block identifier
2	2	Integer	2	Q5STLL	Control block length
4	4	Character	4	Q5STEYEC	Control block eye catcher
8	8	Character	660	Q5STZERO	QMST part cleared on occasion
8	8	Integer	4	NUMTASK	Number of server tasks
12	C	Integer	4	ACTTASK	Number of active server tasks
16	10	Integer	4	CONNCNT	Number of connect requests
20	14	Integer	4	DISCCNT	Number of disconnect requests
24	18	Integer	4	DHIGMAX	Max. request queue depth
28	1C	Integer	4	ABNDCNT	Number of Db2SRV task abends
32	20	Integer	4	REQUCNT	Number of requests requeued
36	24	Integer	4	DEADCNT	Number of deadlock timeouts
40	28	Integer	4	DELECNT	Number of delete requests
44	2C	Integer	4	LISTCNT	Number of list requests
48	30	Integer	4	READCNT	Number of read requests
52	34	Integer	4	UPDTCNT	Number of update requests
56	38	Integer	4	WRITCNT	Number of write requests
60	3C	Integer	4	SCSSEL	SCST (shared-channel-status) selects
64	40	Integer	4	SCSINS	SCST inserts
68	44	Integer	4	SCSUPD	SCST updates
72	48	Integer	4	SCSDEL	SCST deletes

Table 136. Db2 statistics record (Q5ST) (continued)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
76	4C	Integer	4	SSKSEL	SSKT (shared-sync-key) selects
80	50	Integer	4	SSKINS	SSKT inserts
84	54	Integer	4	SSKDEL	SSKT deletes
88	58	Integer	4	SCSBFTS	SCST number of times buffer too small
92	5C	Integer	4	SCSMAXR	SCST maximum rows on query
96	60	Integer	4	* (2)	Reserved
104	68	Character	8	DELETCUW	Cumulative STCK difference - Thread delete
112	70	Character	8	DELETMXW	Maximum STCK difference - Thread delete
120	78	Character	8	DELESCUW	Cumulative STCK difference - SQL delete
128	80	Character	8	DELESMXW	Maximum STCK difference - SQL delete
136	88	Character	8	LISTTCUW	Cumulative STCK difference - Thread list
144	90	Character	8	LISTTMXW	Maximum STCK difference - Thread list
152	98	Character	8	LISTSCUW	Cumulative STCK difference - SQL list
160	A0	Character	8	LISTSMXW	Maximum STCK difference - SQL list
168	A8	Character	8	READTCUW	Cumulative STCK difference - Thread read
17 6	B0	Character	8	READTMXW	Maximum STCK difference - Thread read
184	B8	Character	8	READSCUW	Cumulative STCK difference - SQL read
192	C0	Character	8	READSMXW	Maximum STCK difference - SQL read
200	C8	Character	8	UPDTTCUW	Cumulative STCK difference - Thread update
208	D0	Character	8	UPDTTMXW	Maximum STCK difference - Thread update
216	D8	Character	8	UPDTSCUW	Cumulative STCK difference - SQL update
224	E0	Character	8	UPDTSMXW	Maximum STCK difference - SQL update
232	E8	Character	8	WRITTCUW	Cumulative STCK difference - Thread write
240	F0	Character	8	WRITTMXW	Maximum STCK difference - Thread write
248	F8	Character	8	WRITSCUW	Cumulative STCK difference - SQL write
256	100	Character	8	WRITSMXW	Maximum STCK difference - SQL write
264	108	Character	8	SCSSTCUW	Cumulative STCK difference - Thread select
272	110	Character	8	SCSSTMXW	Maximum STCK difference - Thread select
280	118	Character	8	SCSSSCUW	Cumulative STCK difference - SQL select
288	120	Character	8	SCSSSMXW	Maximum STCK difference - SQL select
296	128	Character	8	SCSITCUW	Cumulative STCK difference - Thread insert
304	130	Character	8	SCSITMXW	Maximum STCK difference - Thread insert
312	138	Character	8	SCSISCUW	Cumulative STCK difference - SQL insert
320	140	Character	8	SCSISMXW	Maximum STCK difference - SQL insert
328	148	Character	8	SCSUTCUW	Cumulative STCK difference - Thread update
336	150	Character	8	SCSUTMXW	Maximum STCK difference - Thread update
344	158	Character	8	SCSUSCUW	Cumulative STCK difference - SQL update
352	160	Character	8	SCSUSMXW	Maximum STCK difference - SQL update
360	168	Character	8	SCSDTCUW	Cumulative STCK difference - Thread delete

Table 136. Db2 statistics record (Q5ST) (continued)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
368	170	Character	8	SCSDTMXW	Maximum STCK difference - Thread delete
376	178	Character	8	SCSDSCUW	Cumulative STCK difference - SQL delete
384	180	Character	8	SCSDSMXW	Maximum STCK difference - SQL delete
392	188	Character	8	SSKSTCUW	Cumulative STCK difference - Thread select
400	190	Character	8	SSKSTMXW	Maximum STCK difference - Thread select
408	198	Character	8	SSKSSCUW	Cumulative STCK difference - SQL select
416	1A0	Character	8	SSKSSMXW	Maximum STCK difference - SQL select
424	1A8	Character	8	SSKITCUW	Cumulative STCK difference - Thread insert
432	1B0	Character	8	SSKITMXW	Maximum STCK difference - Thread insert
440	1B8	Character	8	SSKISCUW	Cumulative STCK difference - SQL insert
448	1C0	Character	8	SSKISMXW	Maximum STCK difference - SQL insert
456	1C8	Character	8	SSKDTCUW	Cumulative STCK difference - Thread delete
464	1D0	Character	8	SSKDTMXW	Maximum STCK difference - Thread delete
472	1D8	Character	8	SSKDSCUW	Cumulative STCK difference - SQL delete
480	1E0	Character	8	SSKDSMXW	Maximum STCK difference - SQL delete
488	1E8	Integer	4	LMSSEL	Number of Db2 BLOB read requests
492	1EC	Integer	4	LMSINS	Number of Db2 BLOB insert requests
496	1F0	Integer	4	LMSUPD	Number of Db2 BLOB update requests
500	1F4	Integer	4	LMSDEL	Number of Db2 BLOB delete requests
504	1F8	Integer	4	LMSLIS	Number of Db2 BLOB list requests
508	IFC	64 bit integer	8	LMSSTCUW	Total elapsed time for all thread read BLOB requests
516	204	64 bit integer	8	LMSSTMXW	Maximum elapsed time for a thread read BLOB request
524	20C	64 bit integer	8	LMSSSCUW	Total elapsed time for all SQL read BLOB requests
532	214	64 bit integer	8	LMSSSMXW	Maximum elapsed time for an SQL read BLOB request
540	21C	64 bit integer	8	LMSITCUW	Total elapsed time for all thread insert BLOB requests
548	224	64 bit integer	8	LMSITMXW	Maximum elapsed time for a thread insert BLOB request
556	22C	64 bit integer	8	LMSISCUW	Total elapsed time for all SQL insert BLOB requests
564	234	64 bit integer	8	LMSISMXW	Maximum elapsed time for an SQL insert BLOB request
572	23C	64 bit integer	8	LMSUTCUW	Total elapsed time for all thread update BLOB requests
580	244	64 bit integer	8	LMSUTMXW	Maximum elapsed time for a thread update BLOB request
588	24C	64 bit integer	8	LMSUSCUW	Total elapsed time for all SQL update BLOB requests

Table 136. Db2 statistics record (Q5ST) (continued)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
596	254	64 bit integer	8	LMSUSMXW	Maximum elapsed time for an SQL update BLOB request
604	25C	64 bit integer	8	LMSDTCUW	Total elapsed time for all thread delete BLOB requests
612	264	64 bit integer	8	LMSDTMXW	Maximum elapsed time for a thread delete BLOB request
620	26C	64 bit integer	8	LMSDSCUW	Total elapsed time for all SQL delete BLOB requests
628	274	64 bit integer	8	LMSDSMXW	Maximum elapsed time for an SQL delete BLOB request
636	27C	64 bit integer	8	LMSLTCUW	Total elapsed time for all thread list BLOB requests
644	284	64 bit integer	8	LMSLTMXW	Maximum elapsed time for a thread list BLOB request
652	28C	64 bit integer	8	LMSLSCUW	Total elapsed time for all SQL list BLOB requests
660	294	64 bit integer	8	LMSLSMXW	Maximum elapsed time for an SQL list BLOB request

The data contains counts for each request type that the Db2 resource manager supports. For these request types, maximum and cumulative elapse times are kept for the following:

- The time spent in the Db2 resource manager as a whole (called the thread time).
- The time that was spent performing the RRSAF and SQL parts of the request (a subset of the thread time called the SQL time).

Information is also provided for:

- The number of server tasks attached.
- The maximum overall request depth against any of the server tasks.
- The number of times any of the server task requests terminated abnormally.

If the abnormal termination count is not zero, a requeue count is provided indicating the number of queued requests that were requeued to other server tasks as a result of the abnormal termination.

If the average thread time is significantly greater that the average SQL time, this might indicate that thread requests are spending an excessive amount of time waiting for a server task to process the SQL part of the request. If this is the case, examine the DHIGMAX field and, if the value is greater than one, consider increasing the number of Db2 server tasks specified in the QSGDATA parameter of the CSQ6SYSP system parameter macro.

Coupling facility manager data records:

Use this topic as a reference to the format of the coupling facility manager data records.

The format of the coupling facility manager statistics record is described in the following table and in assembler macro thlqual. SCSQMACS(CSQDQEST) and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example qest, qestid.

If the queue manager was not started as a member of a queue-sharing group, no data is recorded in this record.

Table 137. Coupling facility statistics record (QEST)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
0	0	Structure	4104	QEST	CF manager statistics
0	0	Bitstring	2	QESTID	Control block identifier
2	2	Integer	2	QESTLL	Control block length
4	4	Character	4	QESTEYEC	Control block eye catcher
8	8	Character	4096	QESTZERO	QEST part cleared on occasion
8	8	Character	64	QESTSTUC (0:63)	Array (one entry per structure)
8	8	Character	12	QESTSTR	Structure name
20	14	Integer	4	QESTSTRN	Structure number
24	18	Integer	4	QESTCSEC	Number of IXLLSTE calls
28	1C	Integer	4	QESTCMEC	Number of IXLLSTM calls
32	20	Character	8	QESTSSTC	Time spent doing IXLLSTE calls
40	28	Character	8	QESTMSTC	Time spent doing IXLLSTM calls
48	30	Integer	4	QESTRSEC	Number of IXLLSTE redrives
52	34	Integer	4	QESTRMEC	Number of IXLLSTM redrives
56	38	Integer	4	QESTSFUL	Number of structure fulls
60	3C	Integer	4	QESTMNUS	Maximum number of entries in use
64	40	Integer	4	QESTMLUS	Maximum number of elements in use
68	44	Character	4	*	Reserved
4104	1008	Character	0	*	End of control block

The data contains information for each coupling facility list structure, including the CSQ_ADMIN structure, that the queue manager could connect to during the statistics interval. The information for each structure includes the following:

- The number of and cumulative elapsed times for IXLLSTE and IXLLSTM requests.
- The number of times a request had to be retried because of a timeout.
- The number of times a 'structure full' condition occurred.

Topic manager data records:

Use this topic as a reference to the format of the topic manager data records.

The format of the Topic manager statistics record is described in the following table and in assembler macro thlqual.SCSQMACS(CSQDQTST) and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example qtst, qtstid.

Table 138. Topic manager statistics record (QTST)

Offset: Dec	Offset: Hex	Type	Len	Name	Description
0	0	Structure	96	QTST	Topic manager statistics
0	0	Bitstring	2	QTSTID	Control block identifier
2	2	Integer	2	QTSTLL	Control block length
4	4	Character	4	TESTEYEC	Control block eye catcher
8	8	Character	88	QTSTZERO	QTST part cleared on occasion
8	8	Integer	4	QTSTSTOT	Total subscription requests
12	0C	Integer	4	QTSTSDUR	Durable subscription requests
16	10	Integer	4	QTSTSHIG (1:3)	Subscription high water mark array (API, ADMIN, PROXY)
28	1C	Integer	4	QTSTSLOW (1:3)	Subscription low water mark array (API, ADMIN, PROXY)
40	28	Integer	4	QTSTSEXP	Subscriptions expired
44	2C	Integer	4	QTSTTMSG	Total messages put to Sub queue
48	30	Integer	4	QTSTSPHW	Single publish subscriber high water mark
52	34	Integer	4	QTSTPTOT (1:3)	Total Publication requests (API, ADMIN, PROXY)
64	40	Integer	4	QTSTPTHI	Total publish high water mark
68	44	Integer	4	QTSTPTLO	Total publish low water mark
72	48	Integer	4	QTSTPNOS	Count of publishes to no subscriber
76	4C	Integer	4	*	Reserved
80	50	Bitstring	8	QTSTETHW	Elapse time HW on publish
88	58	Bitstring	8	QTSTETTO	Elapse time total on publish

Coupling facility manager SMDS data records:

Use this topic as a reference to the format of the coupling facility manager shared message data set (SMDS) data records.

The format of the coupling facility manager shared message data set (SMDS) statistics record is described in assembler macro thlqual.SCSQMACS(CSQDQESD), C header file thlqual.SCSQC370(CSQDSMFC) and in IBM MQ SupportPac MP1B.

The statistics provide information about the utilization of the owned shared message data set, I/O activity for the group of shared message data sets, and SMDS buffer utilization.

If the queue manager was not started as a member of a queue-sharing group, no data is recorded in this record.

Layout of SMF records for the channel initiator:

The layouts of channel accounting data (SMF type 116, subtype 10) and channel initiator statistics data (SMF type 115 , subtype 231 records) are described in this topic.

Processing the SMF data for the CHINIT

The data written to SMF is in the standard triplet format.

Accounting data SMF type 116, subtype 10

There is the standard SMF header.

The triplets are mapped by qws5 in csqdsmfc.h and csqdqws5.macro, and have the following layout:

4 bytes offset to the QWHS
2 bytes length of the QWHS
2 bytes count of the number of instances of QWHS
4 bytes offset to the QCST
2 bytes length of the QCST
2 bytes count of the number of instances of QCST

The QWHS mapped is mapped by csqdqwhs.macro and csqdsmfc.h, and has the following key fields:

- qwhsnsda 1 byte, count of the number of self defining section.
- qwhssmfc 1 bit. If this is on there are multiple SMF records containing information for this interval. If this is off, this is the last or only record.
- Qwhstime 8 bytes in STCK format. The local time of the start of the interval.
- qwhsdurn 8 bytes in STCK format. The duration from the start of the interval to the end of the interval.
- Qwhsstck 8 bytes STCK format. The end of the interval in GMT.

The QCST is mapped by csqdsmfc.h and csqdqcst.macro.

Statistics data SMF type 115, subtype 231

There is the standard SMF header.

The triplets are mapped by qwsx in csqdsmfc.h and csqdqwsx.macro, and have the following layout:

4 bytes offset to the QWHS

2 bytes length of the QWHS
2 bytes count of the number of instances of QWHS

CHINIT Control Information, number of channels. and so on is mapped by csqdsmfc.h and csqdqcct.macro:
4 bytes offset to the QCCT
2 bytes length of the QCCT
2 bytes count of the number of instances of the QCCT

Dispatcher tasks are mapped by csqdsmfc.h and the QCT_DSP structure in the CSQDQCTA macro:
4 bytes offset to the QCT_DSP
2 bytes length of the QCT_DSP
2 bytes count of the number of instances of the QCT_DSP

Adapter tasks are mapped by csqdsmfc.h and the QCT_ADP structure in the CSQDQCTA macro:
4 bytes offset to the QCT_ADP
2 bytes length of the QCT_ADP
2 bytes count of the number of instances of QCT_ADP

SSL tasks are mapped by csqdsmfc.h and the QCT_SSL structure in the CSQDQCTA macro:
4 bytes offset to the QCT_SSL
2 bytes length of the QCT_SSL
2 bytes count of the number of instances of QCT_SSL

DNS task is mapped by csqdsmfc.h and the QCT_DNS structure in the CSQDQCTA macro:
4 bytes offset to the QCT_DNS
2 bytes length of the QCT_DNS
2 bytes count of the number of instances of QCT_DNS

Typically one record contains all the data. If there are a large number of dispatchers, adapters, or SSL tasks, the data is split over more than one record.

If this happens, the count of instances of the dispatchers can be zero, and information about a group of TCBs can be spread across multiple records. For example the number of instances can look like this:

Table 139. Example data

Count	**First record**	**Last record**
QWHS	1	1
QCCT	1	0
QDSP	50	5
QADP	0	10
QSSL	0	3
QDNS	0	1

This example shows that there were 55 dispatcher TCBs within the SMF interval.

The field `qwhs.qwhssmfc` indicates a continuation. If this bit is on, there are multiple SMF records containing information for this interval. If this bit is off, this is the last or only record.

Channel initiator statistics data records:

Use this topic as a reference for channel initiator statistics data records.

The format of the channel initiator statistics data record contains two parts:
- The first part is the channel initiator control information block, described in assembler macro `thlqual.SCSQMACS(CSQDQCCT)`. For further information, see"Channel initiator control information block."
- The second part is the channel initiator task block, described in assembler macro `thlqual.SCSQMACS(CSQDQCTA)`.

  The channel initiator task block contains information about the four types of task within the CHINIT. For further information, see:
  - "Dispatcher tasks" on page 1247
  - "Adapter tasks" on page 1248
  - "Domain Name Server (DNS) task" on page 1249
  - "SSL tasks" on page 1250

  Each task includes:
  - The elapsed time the task was active in the interval (*qcteltm*)
  - How much CPU time was using in the interval (*qctcptm*)
  - Total wait time of this task in the interval (*qctwttm*)
  - The number of requests in the interval (*qctreqn*)

  You can use this information to see how busy the task was, and determine whether you need to add more tasks based on the analysis.

  For SSL and DNS tasks, the duration of the longest request (*qctlgdu*, *qctlsdu*) and the time of day when this occurred (*qctlgdm*, *qctlsdm*) are also included.

  These can be useful to identify when channel requests took a long time. For example, a DNS lookup request going to a server outside of your enterprise taking seconds rather than milliseconds.

  The example accounting data in the following tasks has been formatted using IBM MQ SupportPac MP1B.

Both of the parts are also described in the C programming language header file `thlqual.SCSQC370(CSQDSMFC)`. Note that the field names in C are all in lowercase, for example, *qcct*, *qct_adp*.

Channel initiator control information block:

Use this topic as a reference for the channel initiator control information block.

The channel initiator control information block contains basic information for this CHINIT, including:
- CHINIT job name (*qcctjobn*)
- QSG name if it is in a QSG (*qcctqsgn*)
- Peak number used of current channels (*qcctnocc*)
- Peak number used of active channels (*qcctnoac*)
- MAXCHL - maximum permitted current channels (*qcctmxcc*)
- ACTCHL - maximum permitted active channels (*qcctmxac*)
- TCPCHL - maximum permitted TCP/IP channels (*qcctmxtp*)
- LU62CHL - maximum permitted LU62 channels (*qcctmxlu*)
- Storage used by CHINIT (*qcctstus*)

You can use this information to see if the number of active channels is approaching the configured maximum value. Note that the number of current and active channels are the values when the record was created. So, between the two intervals there might have been more than this number of channels active.

Channel information from SMF data

Here is an example of channel information from SMF data:

```
MVCA,MQPV,2014/03/18,13:00:00,VRM:800,
From 2014/03/18,12:45:00.015222 to 2014/03/18,13:00:00.083630 duration 900.068408 seconds
Peak number used of current channels.......... 1
Peak number used of active channels .......... 1
MAXCHL. Max allowed current channels..........9999
ACTCHL. Max allowed active channels...........9999
TCPCHL. Max allowed TCP/IP channels...........9999
LU62CHL. Max allowed LU62 channels............ 200
Storage used by Chinit........................ 436MB
```

You can monitor the storage usage and see whether the value is trending upwards. If the total used is approaching the total storage available, you might be running out of storage, and so might not be able to support many more channels.

If the numbers of active current channels are tending towards the maximum number of channels, you might need to increase the maximum number of channels.

Dispatcher tasks:

Example data for the dispatcher tasks, and information about how to interpret the data.

Example data

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET"
    ,     ,         ,       ,  Seconds,       ,  uSeconds, uSeconds
   0, DISP,    26587,    0.4,  0.592463,   0.1,        22,      127
   1, DISP,    26963,    0.3,  0.588092,   0.1,        22,      112
   2, DISP,   864329,    2.7,  2.545668,   0.3,         3,       28
   3, DISP,    26875,    0.4,  0.590825,   0.1,        22,      120
   4, DISP,    26874,    0.4,  0.603285,   0.1,        22,      123
Summ, DISP,   971628,    0.8,  4.920332,   0.1,         5,       38
```

The example data shows that there were five dispatchers. A channel is associated with a dispatcher, and the work is distributed across all the dispatchers. This example shows that one dispatcher is processing more requests than other dispatchers. This is normal, as some channels might stop, so the dispatcher is processing fewer channels, and some channels can be busier than others.

- 4.9 seconds of CPU were used by the dispatchers.
- The average request used 5 microseconds of CPU and took 38 microseconds elapsed time.
- A dispatcher is used to send and receive data over a communications network, and this is not usually dependent on external events. The average elapsed time should, therefore, be close to the average CPU time used. If the CHINIT is delayed due to lack of CPU, then the ratio of average Elapsed Time to average CPU time is much larger, compared to when the CHINIT is not delayed for CPU.
- The average CPU used per request depends on the message traffic, for example, bigger messages use more CPU than smaller messages.

The fields are calculated from:

- Duration: qwhs.qwhsdurn
- Requests : qctreqn
- Busy %: qcteltm and duration

- CPU used: qctcptm
- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn
- Average ET: qcteltm and qctreqn

Usually, the number of dispatchers should be less than, or equal to, the number of processors in the LPAR. If you have more dispatchers than processors in the LPAR they might compete for CPU resources. For more information about tuning your system, see SupportPac MP16.

Channels have an affinity to a dispatcher, so you might find that some dispatchers process many more requests than another dispatcher.

You can use the ALTER QMGR CHIDISPS() command to change the number of dispatchers used. Any change comes into effect the next time the CHINIT is started.

Adapter tasks:

Example data for the adapter tasks, and information about how to interpret the data.

Example data

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET"
    ,     ,         ,       ,   Seconds,      ,  uSeconds, uSeconds
   0, ADAP,   470297,   10.2, 41.290670,   4.6,        88,      194
   1, ADAP,    13907,    0.6,  1.589428,   0.2,       114,      365
   2, ADAP,     2517,    0.2,  0.185325,   0.0,        74,      746
   3, ADAP,     1095,    0.1,  0.085774,   0.0,        78,      907
   4, ADAP,      535,    0.1,  0.040743,   0.0,        76,      947
   5, ADAP,      220,    0.0,  0.016228,   0.0,        74,     1175
   6, ADAP,       82,    0.0,  0.005521,   0.0,        67,     1786
   7, ADAP,       80,    0.0,  0.004248,   0.0,        53,     1160
Summ, ADAP,   488733,    1.4, 43.217938,   0.6,        88,      205
```

The fields are calculated from:
- Duration: qwhs.qwhsdurn
- Requests: qctreqn
- Busy %: qcteltm and duration
- CPU used: qctcptm
- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn average
- ET: qcteltm and qctreqn

This example shows that there were eight adapter tasks.

Adapter number 0
- Processed the majority of the requests (470297 out of 488733)
- Was busy 10.2% of the interval
- Used 41.3 seconds of CPU

Overall
The average CPU per request was 88 microseconds of CPU and took 205 microseconds

The adapters process IBM MQ requests. Some of these requests might wait, for example, for log I/O during a commit, so the average Elapsed Time per request has little meaning.

When an IBM MQ request is made the first free adapter task is used.
- If there is at least one adapter that has been little used (less than 1%) busy, you have enough adapters.

- If at least one adapter was not used, you have enough adapters defined.
- If all the adapters were used, you might need to allocate more adapters.
- If all of the adapters were used, and they were all busy for most of the interval, you need to allocate more adapters.

You can use the ALTER QMGR CHIADAPS() command to change the number of adapters used. Any changes come into effect the next time the CHINIT is started.

Attention: If there are too many adapters acting on a small set of queues, you might get contention within the queue manager.

Related information:

ALTER QMGR

Domain Name Server (DNS) task:

Example data for the DNS tasks, and information about how to interpret the data.

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET",  longest,       date,            time
   ,     ,          ,       ,   Seconds,      ,  uSeconds, uSeconds, uSeconds,            ,
   0,  DNS,    14002,    0.0,  0.122578,   0.0,         9,       11,      463, 2014/03/18, 12:56:33.987671
Summ,  DNS,    14002,    0.0,  0.122578,   0.0,         9,       11,      463, 2014/03/18, 12:56:33.987671
```

The CHINIT uses a single DNS task. The example shows that the task processed 14002 requests and on average the request used 9 microseconds of CPU and took 11 microseconds of elapsed time.

The longest DNS request took 463 microseconds elapsed time, and this occurred at 12:56:33 local time.

The fields are calculated from:
- Duration: qwhs.qwhsdurn
- Requests : qctreqn
- Busy %: qcteltm and duration
- CPU used: qctcptm
- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn
- Average ET: qcteltm and qctreqn
- Longest: qctlgdu
- Longest at: qctlgtm

The DNS task can go out of your enterprise to look up the IP address associated with a name. If the average Elapsed time is significantly more than the average CPU time used, you might have some long requests.

If the value of the longest request time is unacceptable you should work with your network team to investigate why you are having long requests. It might be that you have an invalid name in your connections.

If the DNS task is busy for 25% of the duration, consider investigating the cause further.

Note: There are requests to the DNS task that are not DNS lookups, so you might have the number of requests being greater than zero - but no longest request information.

SSL tasks:

Example data for the SSL tasks, and information about how to interpret the data.

Example data

```
Task, Type, Requests, Busy %,  CPU used, CPU %, "avg CPU", "avg ET",  longest,       date,           time
    ,     ,         ,       ,   Seconds,      ,  uSeconds, uSeconds, uSeconds,           ,
   0,  SSL,     3112,    1.2,  0.248538,   0.3,        80,      362,     8864, 2014/03/18, 12:46:40.237697
   1,  SSL,     3070,    1.2,  0.245433,   0.3,        80,      359,     4714, 2014/03/18, 12:46:18.938022
   2,  SSL,     3170,    1.2,  0.255557,   0.3,        81,      362,     7273, 2014/03/18, 12:46:35.358145
   3,  SSL,     3060,    1.2,  0.246542,   0.3,        81,      365,    13164, 2014/03/18, 12:46:44.514045
   4,  SSL,     3120,    1.3,  0.251927,   0.3,        81,      373,    22438, 2014/03/18, 12:46:22.134123
Summ,  SSL,    15532,    1.2,  1.247998,   0.3,        80,      364,    22438, 2014/03/18, 12:46:22.134123
```

This example data shows that the average request took 364 microseconds. The longest request was for SSL task 4, took 22,438 microseconds, and occurred at 12:46:22.134123 local time.

The fields are calculated from:

- Duration: qwhs.qwhsdurn
- Requests : qctreqn
- Busy %: qcteltm and duration
- CPU used: qctcptm
- CPU %: qctcptm and duration
- Average CPU: qctcptm and qctreqn
- Average ET: qcteltm and qctreqn
- Longest: qctlsdu longest at: qctlstm

A running channel is associated with an SSL task, in a similar way that a channel is associated with a dispatcher. The SSL tasks can use the cryptographic coprocessors available to the LPAR. So, the elapsed time can include time spent on a coprocessor. You should monitor the average elapsed time throughout the day. If this time increases significantly during peak periods you should work with your MVS systems programmers, as your coprocessors might be over utilized.

If the SSL tasks are busy for a significant proportion of the interval, increasing the number of SSL tasks might help. If the SSL tasks are waiting for external resources such as a coprocesor, increasing the number of SSL tasks has little effect.

You can use the ALTER QMGR SSLTASKS() command to change the number of SSL tasks used. Any changes come into effect the next time the CHINIT is started.

Related information:
ALTER QMGR

Interpreting IBM MQ accounting data

IBM MQ accounting data is written as SMF type 116 records. Use this topic as a reference to the different types of accounting data records.

IBM MQ accounting information can be collected for the following subtypes:

0 Message manager accounting records (how much processor time was spent processing IBM MQ API calls and the number of **MQPUT** and **MQGET** calls). This information is produced when a named task disconnects from IBM MQ, and so the information contained within the record might cover many hours.

1 Accounting data for each task, at thread and queue level.

2 Additional queue-level accounting data (if the task used more queues than could fit in the subtype 1 record).

10 Accounting data for channels.

Note: Accounting information for specific channels can be enabled or suppressed by the STATCHL channel attribute, and the STATACLS queue manager attribute.

Note that:
- Subtype 0 is produced with trace class(1)
- Subtypes 1 and 2 are produced with trace class(3)
- Subtype 10 is produced with accounting trace class(4)

Subtype

Layout of an SMF type 116 record:

Use this topic as a reference to the format of an SMF type record.

The standard layout for SMF records involves three parts:

SMF header
Provides format, identification, and time and date information about the record itself.

Self-defining section
Defines the location and size of the individual data records within the SMF record.

Data records
The actual data from IBM MQ that you want to analyze.

For more information about SMF record formats, see the *MVS System Management Facilities (SMF)* manual.

The SMF header

Table 140 on page 1252 shows the format of SMF record header (SM116).

Table 140. SMF record header description

Offset: Dec	Offset: Hex	Type	Len	Name	Description	Example
0	0	Structure	28	SM116	SMF record header.	
0	0	Integer	2	SM116LEN	SMF record length.	01A4
2	2		2		Reserved.	
4	4	Integer	1	SM116FLG	System indicator.	5E
5	5	Integer	1	SM116RTY	Record type. The SMF record type, for IBM MQ accounting records this is always 116 (X'74').	74
6	6	Integer	4	SM116TME	Time when SMF moved record.	00356124
10	A	Integer	4	SM116DTE	Date when SMF moved record.	0100223F
14	E	Character	4	SM116SID	z/OS subsystem ID. Defines the z/OS subsystem on which the records were collected.	D4E5F4F1 (MV41)
18	12	Character	4	SM116SSI	IBM MQ subsystem ID.	D4D8F0F7 (MQ07)
22	16	Integer	2	SM116STF	Record subtype.	0000
24	18	Character	3	SM116REL	IBM MQ version.	F6F0F0 (600)
27	1B		1		Reserved.	
28	1C	Character	0	SM116END	End of SMF header and start of self-defining section.	
Note: The (hexadecimal) values in the right-hand column relate to Figure 107 on page 1253.						

Self-defining sections

A self-defining section of an SMF record tells you where to find an accounting record, how long it is, and how many times that type of record is repeated (with different values). The self-defining sections follow the header, at a fixed offset from the start of the SMF record.

Each self-defining section points to accounting related data. Table 141 summarizes the offsets from the start of the SMF record header.

Table 141. Offsets to self-defining sections

Record subtype (SMF116STF)	Source of accounting data	Offset of self-defining section		See...
		Dec	Hex	
All	Common header	28	X'1C'	"Common IBM MQ SMF header" on page 1255
0	Message manager	44	X'2C'	"Message manager data records" on page 1257
1	Thread identification record	36	X'24'	"Thread-level and queue-level data records" on page 1258
1	Thread-level accounting	44	X'2C'	"Thread-level and queue-level data records" on page 1258

Table 141. Offsets to self-defining sections (continued)

Record subtype (SMF116STF)	Source of accounting data	Offset of self-defining section		See...
		Dec	Hex	
1	Queue-level accounting	52	X'34'	"Thread-level and queue-level data records" on page 1258. This section is present only if the WTASWQCT field in the task-related information (WTAS) structure is non-zero.
2	Thread identification record	36	X'24'	"Thread-level and queue-level data records" on page 1258
2	Queue-level accounting	44	X'2C'	"Thread-level and queue-level data records" on page 1258
10	Channel accounting			"Channel accounting data records" on page 1260

Note: Other self-defining sections refer to data for IBM use only.

Each self-defining section is two fullwords long and has this format:

```
ssssssssllllnnnn
```

where:

ssssssss
Fullword containing the offset from start of the SMF record.

llll Halfword giving the length of this data record.

nnnn Halfword giving the number of data records in this SMF record.

Figure 107 shows an example of part of an SMF type 116 record. The numbers in the left-hand column represent the offset, in hexadecimal, from the start of the record. Each line corresponds to sixteen bytes of data, where each byte is two hexadecimal characters, for example 0C. The characters in the right-hand column represent the printable characters for each byte. Non-printable characters are shown by a period (.) character.

In this example, alternate fields in the SMF header are underlined to help you to see them; refer to Table 140 on page 1252 to identify them. The self defining section for one of the message manager accounting data records (at the offset given in Table 141 on page 1252) is shown in **bold**.

```
000000 01A40000 5E740035 61240100 223FD4E5  *....;.../.....MV*
000000 F4F1D4D8 F0F70000 F6F0F000 00000134  *41MQ07..600.....*
000000 00700001 00000054 00B00001 00000104  *................*
000000 00300001 00000000 00000000 00000000  *................*
000000 00000000 00000000 00000000 00000000  *................*
```

Figure 107. Part of an SMF record 116 showing the header and self-defining sections. Part of an SMF record 116 showing the header and self-defining sections.

The self-defining section for the type of message manager accounting data is located at offset X'2C' from the start of the SMF record and contains this information:

- The offset of the message manager accounting data is located X'00000104' bytes from the start of the SMF record.
- This message manager record is X'0030' bytes long.
- There is one record (X'0001').

Note: Always use offsets in the self-defining sections to locate the accounting records.

Processing type 116 SMF records:

Use this topic as a reference to the format of the processing type accounting record.

Any accounting data you collect from SMF must be processed to extract useful information. When you process the data, verify that the records are from IBM MQ and that they are the records you are expecting.

Validate the value of the following fields:
- SM116RTY, the SMF record number = X'74' (116)
- SM116STF, the record subtype, must be 0000, 0001, 0002, or 0010

Reading from the active SMF data sets is not supported. You must use the SMF program IFASMFDP to dump SMF records to a sequential data set so that they can be processed. For more information see "Using System Management Facility" on page 1224.

There is a C sample program called CSQ4SMFD which prints the contents of SMF type 115 and 116 records. The program is provided as source in thlqual.SCSQC37S and in executable format in thlqual.SCSQLOAD. Sample JCL is provided in thlqual.SCSQPROC(CSQ4SMFJ).

You need to update the SMFIN DD card with the name of the SMF dataset. Use the z/OS command '/D SMF' to show the name of the dataset, and you need to update the DUMPOUT DD card with the name for the output dataset.

You also need to specify the START and END times that you require.

The following sample JCL extracts SMF records:

```
//SMFDUMP EXEC PGM=IFASMFDP,REGION=0M
//SYSPRINT DD SYSOUT=
//SMFIN  DD DSN=xxxxxx.MANA,DISP=SHR
//SMFOUT  DD DSN=xxxxxx.SMFOUT,SPACE=(CYL,(1,1)),DISP=(NEW,CATLG)
//SYSIN DD *
INDD(SMFIN,OPTIONS(DUMP))
OUTDD(SMFOUT,TYPE(116))
OUTDD(SMFOUT,TYPE(115))
START(1159) END(1210)
/*
```

Common IBM MQ SMF header:

Use this topic as a reference to the common IBM MQ SMF header type accounting record.

The format of this record is described in Table 142 and in assembler macros thlqual.SCSQMACS(CSQDQWHS) and thlqual.SCSQMACS(CSQDQWHC), and C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lowercase, for example qwhs, qwhsnsda.

Details of the structures and fields can be found in IBM MQ supportpac MP1B.

The QWHS data includes the subsystem name. For subtype 1 records, it also shows whether there are queue-level accounting records present. If the QWHSNSDA field is 3 or less, there are not, and the corresponding self-defining section (at offset X'34') is not set.

The QWHC data gives you information about the user (for example, the user ID (QWHCAID) and the type of application (QWHCATYP)). The QWHC section is completed only for subtype 0 records. The equivalent information is present in the thread identification record for subtype 1 and 2 records.

Table 142. Structure of the common IBM MQ SMF header record QWHS

Offset: Dec	Offset: Hex	Type	Length	Name	Description
0	0	Structure	128	QWHS	
0	0		6		Reserved
6	6	Character	1	QWHSNSDA	Number of self defining sections in the SMF records
7	7		5		Reserved
12	C	Character	4	QWHSSSID	Subsystem name
16	10		24		Reserved
40	28	Character	8	QWHCAID	User ID associated with the z/OS job
48	30	Character	12	QWHCCV	Thread cross reference
60	3C	Character	8	QWHCCN	Connection name
68	44		8		Reserved
76	4C	Character	8	QWHCOPID	User ID associated with the transaction
84	54	Integer	4	QWHCATYP	Type of connecting system (1=CICS, 2=Batch or TSO, 3=IMS control region, 4=IMS MPP or BMP, 5=Command server, 6=Channel initiator, 7=RRS Batch)
88	58	Character	22	QWHCTOKN	Accounting token set to the z/OS accounting information for the user
110	6E	Character	16	QWHCNID	Network identifier
126	7E		2		Reserved

Combining CICS and IBM MQ performance data:

Use this topic as a reference to the combination of IBM MQ and CICS performance data.

The common IBM MQ SMF header type accounting record section, QWHCTOKN, is used to correlate CICS type 110 SMF records with IBM MQ type 116 SMF records.

CICS generates an LU6.2 unit-of-work token, for each CICS task. The token is used to generate an accounting token that is written to QWHCTOKN in the correlation header of subtype zero records.

Details are also written to the WTIDACCT section in subtype 1 and 2 records. The accounting token enables correlation between CICS and IBM MQ performance data for a transaction.

Thread cross reference data:

Use this topic as a reference to the format of the thread cross reference type accounting record.

The interpretation of the data in the thread cross reference (QWHCCV) field varies. This depends on what the data relates to:
- CICS connections (QWHCATYP=1) - see Table 143
- IMS connections (QWHCATYP=3 or 4) - see Table 144
- Batch connections (QWHCATYP=2 or 7) - this field consists of binary zeros
- Others - no meaningful data

Table 143. Structure of the thread cross reference for a CICS system

Offset: Dec	Offset: Hex	Type	Length	Description
48	30	Character	4	CICS thread number.
52	34	Character	4	CICS transaction name.
56	38	Integer	4	CICS task number.

Some entries contain blank characters. These apply to the task, rather than to a specific transaction.

Table 144. Structure of the thread cross reference for an IMS system

Offset: Dec	Offset: Hex	Type	Length	Description
48	30	Character	4	IMS partition specification table (PST) region identifier.
52	34	Character	8	IMS program specification block (PSB) name.

Message manager data records:

Use this topic as a reference to the format of the message manager accounting records.

The message manager is the component of IBM MQ that processes all API requests. The format of the message manager accounting records is described in assembler macro thlqual.SCSQMACS(CSQDQMAC).

The QMAC data gives you information about the processor time spent processing IBM MQ calls, and counts of the number of **MQPUT** and **MQGET** requests for messages of different sizes.

Note: A single IMS application might write two SMF records. In this case, add the figures from both records to provide the correct totals for the IMS application.

Records containing zero processor time

Records are sometimes produced that contain zero processor time in the QMACCPUT field. These records occur when long running tasks identified to IBM MQ either terminate or are prompted to output accounting records by accounting trace being stopped. Such tasks exist in the CICS adapter and in the channel initiator (for distributed queuing). The number of these tasks with zero processor time depends upon how much activity there has been in the system:

- For the CICS adapter, this can result in up to nine records with zero processor time.
- For the channel initiator, the number of records with zero processor time can be up to the sum of `Adapters + Dispatchers + 6`, as defined in the queue manager attributes.

These records reflect the amount of work done under the task, and can be ignored.

Sample subtype zero accounting record:

Use this topic as a reference to the format of the subtype zero accounting records.

Figure 108 shows a type 116, subtype zero SMF record. In this figure, the SMF record header and the QMAC accounting data record are underlined. The self-defining sections are in bold.

```
000000  01A40000 5E740035 61240100 223FD4E5   *....;.../.....MV*
000010  F4F1D4D8 F0F70000 F6F0F000 00000134    *41MQ07..600.....*
000020  00700001 00000054 00B00001 00000104   *................*
000030  00300001 00000000 00000000 00000000   *................*
000040  00000000 00000000 00000000 00000000   *................*
000050  00000000 B478AB43 9C6C2280 B478AB47   *.........%......*
000060  9DB47E02 00000000 04C0F631 00000001   *..=......{6.....*
000070  9880E72D 00000000 014D9540 00000000   *..X......(. ....*
000080  08480C80 00000010 40404040 40404040   *........        *
000090  00000000 00000000 00000051 00000000   *................*
0000A0  00000000 00000000 00000000 00000000   *................*
0000B0  00000000 00000000 00000000 00000000   *................*
0000C0  00000000 00000000 00000000 00000000   *................*
0000D0  00000000 00000000 00000000 00000000   *................*
0000E0  00000000 00000000 00000000 00000000   *................*
0000F0  00000000 00000000 00000000 00000000   *................*
000100  00000000 D4140030 D8D4C1C3 00000000   *....M...QMAC....*
000110  689C738D 00000050 00000000 00000050   *.......&.......&*
000120  0000000A 00000000 00000000 00000000   *................*
000130  00000000 0024011A 00030710 02DAACF0   *...............0*
```

Figure 108. Example SMF type 116, subtype zero record

Thread-level and queue-level data records:

Use this topic as a reference to the format of the thread-level and queue-level accounting records.

Thread level accounting records are collected for each task using IBM MQ. For each task, a thread-level accounting data record is written to the SMF when the task finishes. For a long running task, data is also written at the statistics interval set by the STATIME parameter of the CSQ6SYSP system parameter macro (or by the system SMF statistics broadcast), provided that the task was running the previous time statistics were gathered. In addition, accounting information is gathered about each queue that the task opens. A queue-level accounting record is written for each queue that the task has used since the thread-level accounting record was last written.

Thread-level and queue-level accounting records are produced if you specify class 3 when you start the accounting trace.

The thread level accounting information is written to an SMF type 116, subtype 1 record, and is followed by queue-level records. If the task opened many queues, further queue information is written to one or more SMF type 116 subtype 2 records. A thread identification control block is included in each subtype 1 and 2 record to enable you to relate each record to the correct task. Typically, the maximum number of queue-level records in each SMF record is about 45.

The format of the thread-level accounting record is described in assembler macro thlqual.SCSQMACS(CSQDWTAS). The format of the queue-level accounting record is described in assembler macro thlqual.SCSQMACS(CSQDWQ). The format of the thread identification record is described in assembler macro thlqual.SCSQMACS(CSQDWTID). All these records are also described in C header file thlqual.SCSQC370(CSQDSMFC). The field names in C are all in lower case, for example wtas, wtasshex.

Meaning of the channel names:

Use this topic as a reference to the meaning of channel names.

The channel name in the WTID is constructed as shown in the following example. In this example a sender channel exists from queue manager QM1 to queue manager QM2.

Table 145. Meaning of channel names

Field name	Meaning	Example
For queue manager QM1 the sender channel has the following fields set:		
WTIDCCN	The job name	QM1CHIN
WTIDCHL	The channel name	QM1.QM2
WTIDCHLC	This is defined in the CONNAME of the channel	WINMVS2B(2162)
For queue manager QM2 the receiver channel has the following fields set:		
WTIDCCN	The job name	QM2CHIN
WTIDCHL	The channel name	QM1.QM2
WTIDCHLC	Where the channel came from	9.20.101.14

Sample subtype 1 and subtype 2 records:

Use this topic as a reference to the format of the subtype 1 and subtype 2 accounting records.

Figure 109 and Figure 110 show examples of SMF type 116, subtype 1 and subtype 2 records. These two accounting records were created for a batch job that opened 80 queues. Because many queues were opened, a subtype 2 record was required to contain all the information produced.

```
000000  703C0000 5E74002D 983B0100 229FD4E5  *....;.........MV*
000010  F4F1D4D8 F0F70001 F6F0F000 00006FCC  *41MQ07..600...?.*
000020  00700001 0000003C 00D00001 0000010C  *.........}......*
000030  02C00001 000003CC 02400030 F70000D0  *.{....... ..7..}*
000040  E6E3C9C4 00000000 00000000 00000040  *WTID........... *
.
.
.
000100  00000000 00000000 7F4A4BB8 F70102C0  *........"...7..{*
000110  E6E3C1E2 B4802373 0BF07885 7F4AE718  *WTAS.....0..".X.*
```

Figure 109. Example SMF type 116, subtype 1 record. This record contains a CSQDWTID control block, the CSQDWTAS control block, and the first set of CSQDWQST control blocks.

The first self-defining section starts at X'24' and is **bold** in the example; X'0000003C' is the offset to the WTID data record, X'00D0' is the length of the WTID record, and X'0001' is the number of WTID records.

The second self-defining section starts at X'2C' and is in *italic* ; X'0000010C' is the offset to the WTAS data record, X'02C0' is the length of the WTAS record, and X'0001' is the number of WTAS records.

The third self-defining section starts at X'34' and is **bold** in the example; X'000003CC' is the offset to the first WQST data record, X'0240' is the length of the WQST record, and X'0030' is the number of WQST records.

Figure 110 shows an example of an SMF type 116, subtype 2 record.

```
000000  49740000 5E74002D 983B0100 229FD4E5  *....;.........MV*
000010  F4F1D4D8 F0F70002 F6F0F000 00004904  *41MQ07..600.....*
000020  00700001 00000034 00D00001 00000104  *.........}......*
000030  02400020 F70000D0 E6E3C9C4 00000002  *. ..7..}WTID....*
.
.
.
000100  7F4A4BB8 F7020240 E6D8E2E3 00000001  *"...7.. WQST....*
```

Figure 110. Example SMF type 116, subtype 2 record. This record contains a CSQDWTID control block and the remaining CSQDWQST control blocks.

The first self-defining section starts at X'24' and is **bold** in the example; X'00000034' is the offset to the WTID data record, X'00D0' is the length of the WTID record, and X'0001' is the number of WTID records.

The second self-defining section starts at X'2C' and is in *italic* ; X'00000104' is the offset to the first WQST data record, X'0240' is the length of the WQST record, and X'0020' is the number of WQST records.

Figure 111 on page 1260 shows an example of an SMF type 116, subtype 1 record where no queues have been opened and there are consequently no self-defining sections for WQST records..

```
000000          5E740039 4E9B0104 344FD4E5  *    .........|MV*
000010  F4F1D4D8 F0F70001 F6F0F000 000003DC  *41MQ07..600.....*
000020  00800001 00000034 00D00001 00000104  *................*
000030  02D80001 F70000D0 E6E3C9C4 00000002  *.Q..7...WTID....*
000040  C1F8C5C1 C4C5D740 C1F8C5C1 C4C54040  *A8EADEP A8EADE  *
000050  40404040 40404040 00000000 00000000  *        ........*
000060  40404040 40404040 4040               *          *
```

Figure 111. Example SMF type 116, subtype 1 record with no WQST data records

The first self-defining section starts at X'24' and is **bold** in the example; X'00000034' is the offset to the WTID data record, X'00D0' is the length of the WTID record, and X'0001' is the number of WTID records.

The second self-defining section starts at X'2C' and is in *italic* ; X'0000010C' is the offset to the WTAS data record, X'02D8' is the length of the WTAS record, and X'0001' is the number of WTAS records.

There is no self-defining section describing a WQST data record, equivalent to the third self-defining section in Figure 109 on page 1259.

Channel accounting data records:

Use this topic as a reference for channel accounting data records.

The format of the channel accounting data record is described in assembler macro `thlqual.SCSQMACS(CSQDQCST)`. The format is also described in the C programming language header file `thlqual.SCSQC370(CSQDSMFC)`. Note that the field names in C are all in lowercase, for example, *qcst*.

The channel accounting data gives you information about the status and statistics of each channel instance, including:
- Average network time (*qcstntav*)
- Average time on exit (*qcstetav*)
- Channel batch data limit (*qcstcbdl*)
- Channel batch interval (*qcstcbit*)
- Channel batch size (*qcstcbsz*)
- Channel dispatcher number (*qcstdspn*)
- Channel disposition (*qcstchdp*)
- Channel name (*qcstchnm*)
- Channel state (*qcstchst*)
- Channel started time (*qcststrt*)
- Channel status collected time (*qcstcltm*)
- Channel stopped time (*qcstludt*)
- Channel type (*qcstchty*)
- Common name (CN) from SSLCERTI (*qcstslcn*)
- Compression rate (*qcstcpra*)
- Connection name (*qcstcnnm*)
- Current shared conversations (*qcstcscv*)
- DNS resolution time (*qcstdnrt*)
- Effective value of STATCHL parameter (*qcststcl*)
- Last message time (*qcstlmst*)
- Maximum network time (qcstntmx)

- Maximum time on exit (*qcstetmx*)
- Minimum network time (qcstntmn)
- Minimum time on exit (*qcstetmn*)
- Name of the remote queue manager or application (*qcstrqmn*)
- Number of batches (*qcstbatc*)
- Number of bytes for message data (*qcstnbyt*)
- Number of bytes for persistent message data (*qcstnpby*)
- Number of bytes received for both message data and control information (*qcstbyrc*)
- Number of bytes sent for both message data and control information (*qcstbyst*)
- Number of full batches (*qcstfuba*)
- Number of messages, or number of MQI calls (*qcstnmsg*)
- Number of persistent messages (*qcstnpmg*)
- Number of put retries (*qcstptrc*)
- Number of transmission queue becoming empty (*qcstqetc*)
- Number of transmission buffers received (qcstbfrc)
- Number of transmission buffers sent (*qcstbfst*)
- Serial number from SSLPEER (*qcstslsn*)
- SSL CipherSpec (zero means SSL not used) (*qcstslcs*)
- The date and time of maximum network time (*qcstntdt*)
- The date and time of maximum time on exit (*qcstetdt*)

You can use this information to see the throughput of a channel, if the actual batches are approaching the limit, the latency of the network, information about the remote end, performance of user exit, and so on.

Here is an example of the channel accounting data which has been formatted with IBM MQ SupportPac MP1B.

The fields available are based on the display channel status command (DIS CHS) and channel statistics by IBM MQ on platforms except z/OS, with some additional fields.

```
The data and time of the start and end of the record in local time, and the duration
SMF interval start       2014/03/26,02:30:00
SMF interval end         2014/03/26,02:45:00
SMF interval duration    899.997759 seconds

Information about the channel

Connection name       9.20.4.159
Channel disp          PRIVATE
Channel type          RECEIVER
Channel status        CLOSING
Channel STATCHL       HIGH


Start date & time           2014/03/26,02:44:58
Channel status collect time  2014/03/26,02:45:00
Last status changed         1900/01/01,00:00:00
Last msg time               2014/03/26,02:44:59


Batch size                     50
Messages/batch                 3.3
Number of messages             1,102
Number of persistent messages   1,102
Number of batches              335
Number of full batches         0
Number of partial batches      335
Buffers sent                   337
Buffers received               1,272
Message data                   5,038,344   4 MB
Persistent message data        5,038,344   4 MB
Non persistent message data    0   0 B
Total bytes sent               9,852   9 KB
Total bytes received           5,043,520   4 MB
Bytes received/Batch           15,055  14 KB
Bytes sent/Batch               29  29 B
Batches/Second                 1
Bytes received/message         4,576   4 KB
Bytes sent/message             8   8 B
Bytes received/second          28,019  27 KB/sec
Bytes sent/second              54  54 B/sec
Compression rate               0

The name of the queue manager at the remote end of the connection
Remote qmgr/app           MQPH
Put retry count           0
```

Tuning your IBM MQ network

Use the tuning tips in this section to help improve the performance of your queue manager network.

Tuning client and server connection channels

The default settings for client and server connection channels changed in Version 7.0 to use shared conversations. Performance enhancements for distributed severs were then introduced in Version 8.0. To benefit from the new features that were introduced alongside shared conversations, without the performance impact on the distributed server, set **SHARECNV** to 1 on your Version 8.0 server connection channels.

From Version 7.0, each channel is defined by default to run up to 10 client conversations per channel instance. Before Version 7.0, each conversation was allocated to a different channel instance. The enhancements added in Version 7.0 also include the following features:

- Bi-directional heartbeats
- Administrator stop-quiesce
- Read-ahead
- Asynchronous-consume by client applications

For some configurations, using shared conversations brings significant benefits. However, for distributed servers, processing messages on channels that use the default configuration of 10 shared conversations is on average 15% slower than on channels that do not use shared conversations. On an MQI channel instance that is sharing conversations, all of the conversations on a socket are received by the same thread. If the conversations sharing a socket are all busy, the conversational threads contend with one another to use the receiving thread. The contention causes delays, and in this situation using a smaller number of shared conversations is better.

You use the **SHARECNV** parameter to specify the maximum number of conversations to be shared over a particular TCP/IP client channel instance. For details of all possible values, and of the new features added in Version 7.0, see MQI client: Default behavior of client-connection and server-connection. If you do not need shared conversations, there are two settings that give best performance in Version 8.0:

- `SHARECNV(1)`. Use this setting whenever possible. It eliminates contention to use the receiving thread, and your client applications can take advantage of the new features added in Version 7.0. For this setting, distributed server performance is significantly improved in Version 8.0. The performance improvements apply to Version 8.0 client applications that issue non read ahead synchronous get wait

calls; for example C client MQGET wait calls. When these client applications are connected, the distributed server uses less threads and less memory and the throughput is increased.
- `SHAREcnv(0)`. The channel instance behaves exactly as if it was a Version 6.0 server or client connection channel, and you do not get the extra features such as bi-directional heartbeats that are available when you set **SHARECNV** to 1 or greater. Use a value of 0 only if you have existing client applications that do not run correctly when you set **SHARECNV** to 1 or greater.

Note: If a server has clients connected to it that are sharing conversations over a socket, and you decrease the shared conversations setting from `SHARECNV(10)` to `SHARECNV(1)`, this has the following effects:
- Increased socket usage on the server.
- Increased channel instances on the server.

In this case, you might also choose to increase the settings for **MaxChannels** and **MaxActiveChannels**.

For consistency with previous releases the default SVRCONN channel has not been updated, so you need explicitly to set **SHARECNV** to 1 or 0.

Related information:

MQI client: Default behavior of client-connection and server-connection

Tuning distributed publish/subscribe networks

Use the tuning tips in this section to help improve the performance of your IBM MQ distributed publish/subscribe clusters and hierarchies.

Related concepts:

"Monitoring clusters" on page 1218
Within a cluster you can monitor application messages, control messages, and logs. There are special monitoring ocnsiderations when the cluster load balances between two or more instances of a queue.

Direct routed publish/subscribe cluster performance

In direct routed publish/subscribe clusters, information such as clustered topics and proxy subscriptions is pushed to all members of the cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This process can create a significant additional load on the system. To reduce the effect of cluster management on performance you can perform updates at off-peak times, define a much smaller subset of queue managers involved in publish/subscribe and make that an "overlapping" cluster, or switch to using topic host routing.

There are two sources of workload on a queue manager in a publish/subscribe cluster:
- Directly handling messages for application programs.
- Handling messages and channels needed to manage the cluster.

In a typical point-to-point cluster, the cluster system workload is largely limited to information explicitly requested by members of the cluster as required. Therefore in anything other than a very large point-to-point cluster, for example one which contains thousands of queue managers, you can largely discount the performance effect of managing the cluster. However, in a direct routed publish/subscribe cluster, information such as clustered topics, queue manager membership and proxy subscriptions is pushed to all members of the cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This can create a significant additional load on the system. Therefore you need to consider the effect of cluster management on queue manager performance, both in its timing, and its size.

Performance characteristics of direct routed clusters

Compare a point-to-point cluster with a direct routed publish/subscribe cluster in respect of the core management tasks.

First, a point to point cluster:

1. When a new cluster queue is defined, the destination information is pushed to the full repository queue managers, and only sent to other cluster members when they first reference a cluster queue (for example, when an application attempts to open it). This information is then cached locally by the queue manager to remove the need to remotely retrieve the information each time the queue is accessed.
2. Adding a queue manager to a cluster does not directly affect the load on other queue managers. Information about the new queue manager is pushed to the full repositories, but channels to the new queue manager from other queue managers in the cluster are only created and started when traffic begins to flow to or from the new queue manager.

In summary, the load on a queue manager in a point-to-point cluster is related to the message traffic it handles for application programs and is not directly related to the size of the cluster.

Second, a direct routed publish/subscribe cluster:

1. When a new cluster topic is defined, the information is pushed to the full repository queue managers, and from there directly to all members of the cluster, causing channels to be started to each member of the cluster from the full repositories if not already started. If this is the first direct clustered topic, each queue manager member is sent information about all other queue manager members in the cluster.
2. When a subscription is created to a cluster topic on a new topic string, the information is pushed directly from that queue manager to all other members of the cluster immediately, causing channels to be started to each member of the cluster from that queue manager if not already started.
3. When a new queue manager joins an existing cluster, information about all clustered topics (and all queue manager members if a direct cluster topic is defined) is pushed to the new queue manager from the full repository queue managers. The new queue manager then synchronizes knowledge of all subscriptions to cluster topics in the cluster with all members of the cluster.

In summary, cluster management load at any queue manager in a direct routed publish/subscribe cluster grows with the number of queue managers, clustered topics, and changes to subscriptions on different topic strings within the cluster, irrespective of the local use of those cluster topics on each queue manager.

In a large cluster, or one where the rate of change of subscriptions is high, this level of cluster management can be a significant overhead across all queue managers.

Reducing the effect of direct routed publish/subscribe on performance

To reduce the effect of cluster management on the performance of a direct routed publish/subscribe cluster, consider the following options:

- Perform cluster, topic, and subscription updates at off-peak times of the day.
- Define a much smaller subset of queue managers involved in publish/subscribe, and make that an "overlapping" cluster. This cluster is then the cluster where cluster topics are defined. Although some queue managers are now in two clusters, the overall effect of publish/subscribe is reduced:
  - The size of the publish/subscribe cluster is smaller.
  - Queue managers not in the publish/subscribe cluster are much less affected by cluster management traffic.

If the previous options do not adequately resolve your performance issues, consider using a *topic host routed* publish/subscribe cluster instead. For a detailed comparison of direct routing and topic host routing in publish/subscribe clusters, see Designing publish/subscribe clusters.

Topic host routed publish/subscribe cluster performance

A topic host routed publish/subscribe cluster gives you precise control over which queue managers host each topic. These topic hosts become the *routing* queue managers for that branch of the topic tree. Moreover, queue managers without subscriptions or publishers have no need to connect with the topic hosts. This configuration can significantly reduce the number of connections between queue managers in the cluster, and the amount of information that is passed between queue managers.

A topic host routed publish/subscribe cluster behaves as follows:

- Topics are manually defined on individual *topic host* queue managers in the cluster.
- When a subscription is made on a cluster queue manager, proxy subscriptions are created only on the topic hosts.
- When an application publishes information to a topic, the receiving queue manager forwards the publication to a queue manager that hosts the topic. The topic host then sends the publication to all queue managers in the cluster that have valid subscriptions to the topic.

For a more detailed introduction to topic host routing, see Topic host routing in clusters.

For many configurations, topic host routing is a more appropriate topology than *direct routing* because it provides the following benefits:

- Improved scalability of larger clusters. Only the topic host queue managers need to be able to connect to all other queue managers in the cluster. Therefore, there are fewer channels between queue managers, and there is less inter-queue manager publish/subscribe administrative traffic than for direct routing. When subscriptions change on a queue manager, only the topic host queue managers need to be informed.
- More control over the physical configuration. With direct routing, all queue managers assume all roles, and therefore all need to be equally capable. With topic host routing, you explicitly choose the topic host queue managers. Therefore, you can ensure that those queue managers are running on adequate equipment, and you can use less powerful systems for the other queue managers.

However, topic host routing also imposes certain constraints upon your system:

- System configuration and maintenance require more planning than for direct routing. You need to decide which points to cluster in the topic tree, and the location of the topic definitions in the cluster.
- Just as for direct routed topics, when a new topic host routed topic is defined, the information is pushed to the full repository queue managers, and from there direct to all members of the cluster. This event causes channels to be started to each member of the cluster from the full repositories if not already started.
- Publications are always sent to a host queue manager from a non-host queue manager, even if there are no subscriptions in the cluster. Therefore, you should use routed topics when subscriptions are typically expected to exist, or when the overhead of global connectivity and knowledge is greater than the risk of extra publication traffic.
- Messages that are published on non-host queue managers do not go direct to the queue manager that hosts the subscription, they are always routed through a topic host queue manager. This approach can increase the total overhead to the cluster, and increase message latency and reduce performance.

  Note: For certain configurations, you can usefully remove this constraint as described in Topic host routing using centralized publishers or subscribers.
- Using a single topic host queue manager introduces a single point of failure for all messages that are published to a topic. You can remove this single point of failure by defining multiple topic hosts. However, having multiple hosts affects the order of published messages as received by subscriptions.
- Extra message load is incurred by topic host queue managers, because publication traffic from multiple queue managers needs to be processed by them. This load can be lessened: Either use multiple topic hosts for a single topic (in which case message ordering is not maintained), or use different queue managers to host routed topics for different branches of the topic tree.

Topic host routing with centralized publishers or subscribers

To remove the extra "hop" incurred when publications are always routed to subscriptions through a topic host queue manager, configure the publishers or the subscriptions on the same queue manager that hosts the topic. This approach brings maximum performance benefits in the following two cases:

- Topics with many publishers and few subscriptions. In this case, host the subscriptions on the topic host queue manager.
- Topics with few publishers and many subscriptions. In this case, host the publishers on the topic host queue manager.

The following figure shows a topic host queue manager that also hosts the subscriptions. This approach removes the extra "hop" between the publisher and the subscriber, and reduces unnecessary sharing of subscription knowledge across all members of the cluster:

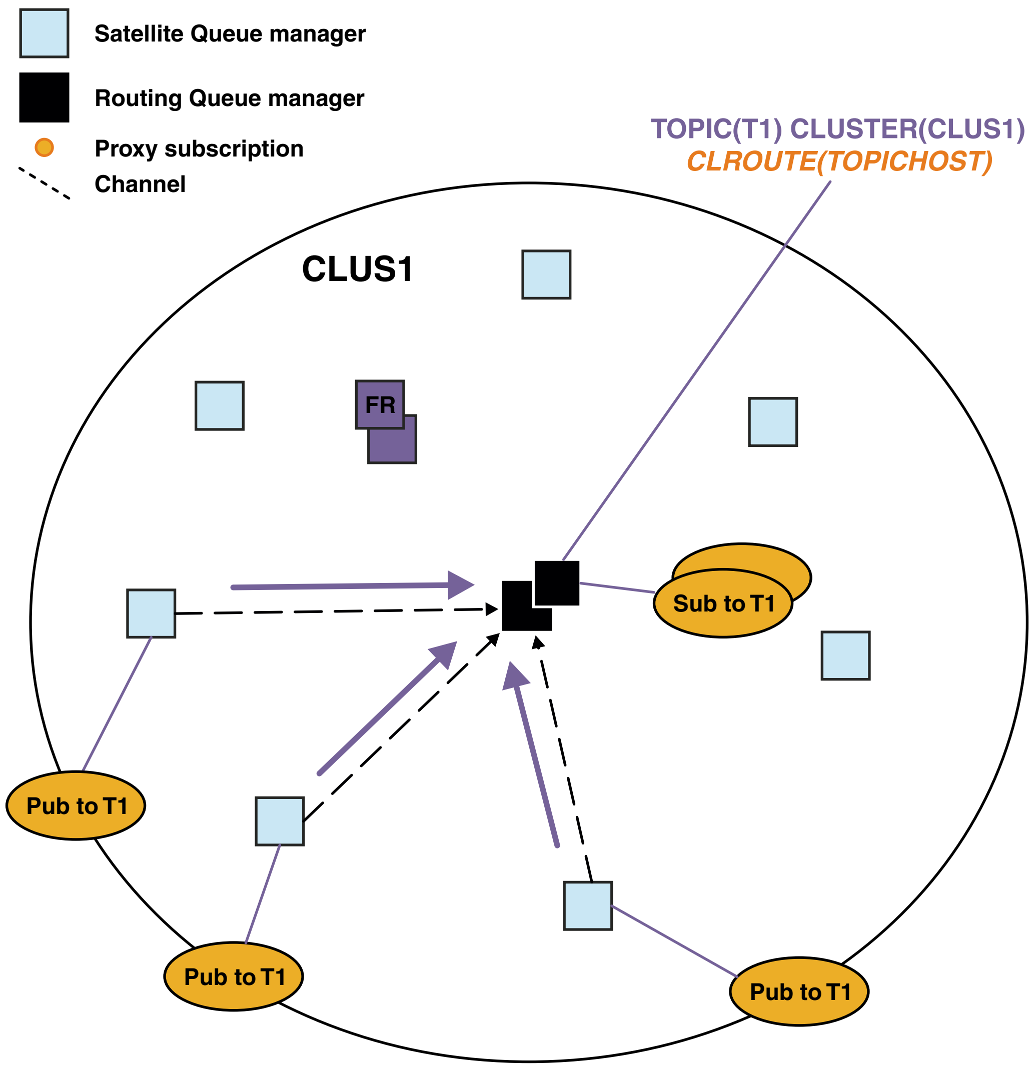


Figure 112. Hosting subscriptions on a topic host queue manager

The following figure shows a topic host queue manager that also hosts the publishers. This approach removes the extra "hop" between the publisher and the subscriber, and reduces unnecessary sharing of subscription knowledge across all members of the cluster:

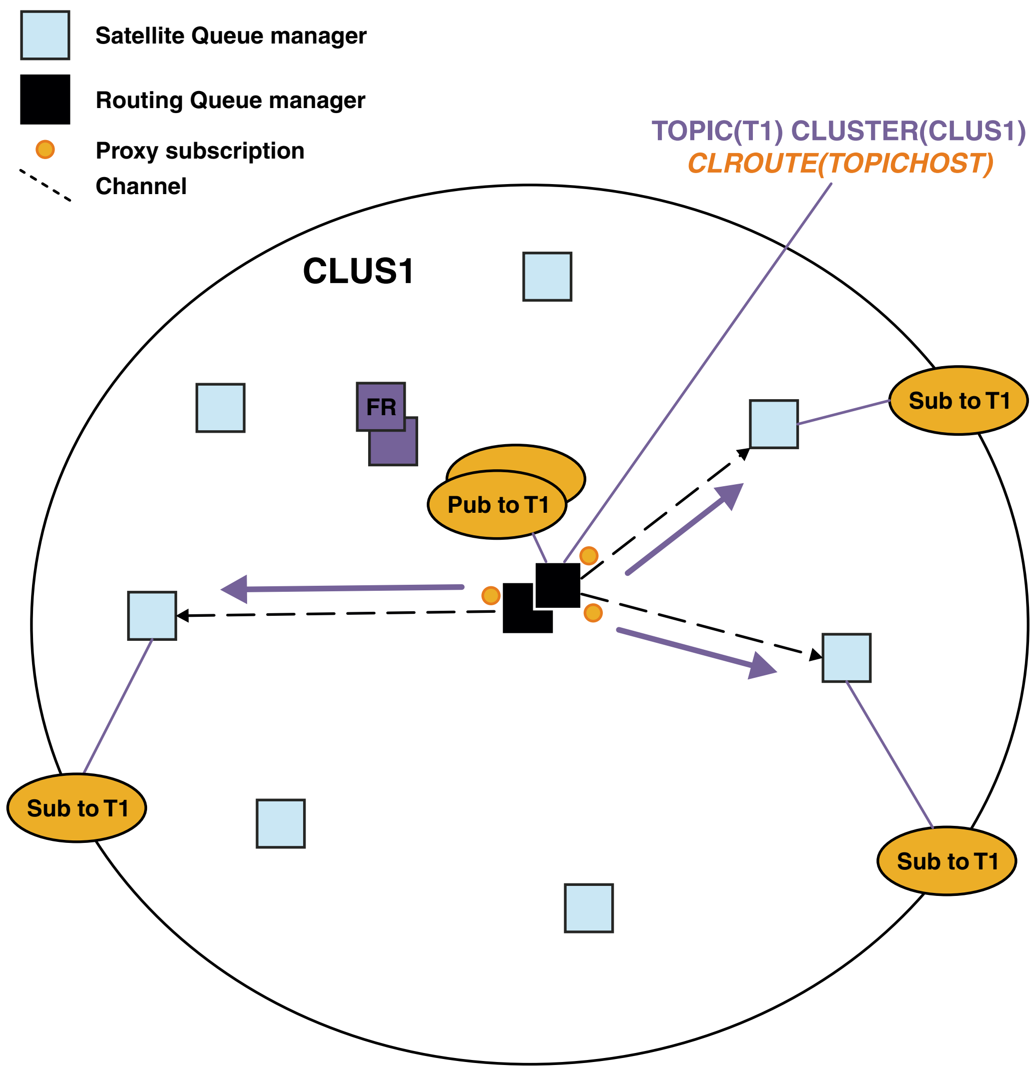


Figure 113. Hosting publications on a topic host queue manager

Balancing producers and consumers in publish/subscribe networks

An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

In a point-to-point messaging topology, the relationship between message consumers and message producers is readily understood. You can get estimates of message production and consumption, queue by queue, channel by channel. If there is a lack of balance, the bottlenecks are readily identified and then remedied.

It is harder to work out whether publishers and subscribers are balanced in a publish/subscribe topology. Start from each subscription, and work back to the queue managers having publishers on the topic. Calculate the number of publications flowing to each subscriber from each queue manager.

Each publication that matches a subscription on a remote queue manager (based on proxy subscriptions) is put to a transmission queue. If multiple remote queue managers have proxy subscriptions for that publication, multiple copies of the message are put to a transmission queue, each targeted for a different sender channel.

In a publish/subscribe cluster, those publications are targeted at the SYSTEM.INTER.QMGR.PUBS queue on the remote queue managers that host the subscriptions. In a hierarchy, each publication is targeted at the SYSTEM.BROKER.DEFAULT.STREAM queue, or any other stream queues listed in the SYSTEM.QPUBSUB.QUEUE.NAMELIST on the remote queue managers. Each queue manager processes messages arriving on that queue and delivers them to the correct subscriptions on that queue manager.

For this reason, monitor the load at the following points where bottlenecks might arise:

- Monitor the load at the individual subscription queues.
  - This bottleneck implies that the subscribing application is not consuming the publications as quick as they are being published.
- Monitor the load at the SYSTEM.INTER.QMGR.PUBS queue or the stream queues.
  - This bottleneck implies that the queue manager is receiving publications from one or more remote queue managers faster than it can distribute them to the local subscriptions.
  - When seen on a topic host queue manager when using topic host routing in a cluster, consider making additional queue managers topic hosts, allowing the publication workload to be balanced across them. However, this will affect the message ordering across publications. See Topic host routing using multiple topic hosts for a single topic.
- Monitor the load at the channels between the publishing queue manager and the subscribing queue managers, which are fed by the transmission queues on the publishing queue manager.
  - This bottleneck implies that either one or more channels is not running, or messages are being published to the local queue manager faster than the channels can deliver them to the remote queue manager.
  - When you use a publish/subscribe cluster, consider defining additional cluster receiver channels on the target queue manager. This allows the publication workload to be balanced across them. However, this affects the message ordering across publications. Also consider moving to a multiple cluster transmission queue configuration, because this can improve performance in certain circumstances.
- If the publishing application is using a queued publish/subscribe interface, monitor the load at (a) the SYSTEM.BROKER.DEFAULT.STREAM queue, and any other stream queues listed in the SYSTEM.QPUBSUB.QUEUE.NAMELIST ; and (b) the SYSTEM.BROKER.DEFAULT.SUBPOINT queue, and any other subpoint queues listed in the SYSTEM.QPUBSUB.SUBPOINT.NAMELIST .
  - This bottleneck implies that messages are being put by local publishing applications faster than the local queue manager can process the messages.

Related concepts:

"Monitoring clusters" on page 1218
Within a cluster you can monitor application messages, control messages, and logs. There are special monitoring ocnsiderations when the cluster load balances between two or more instances of a queue.

Subscription performance in publish/subscribe networks

Distributed publish/subscribe in IBM MQ works by propagating knowledge of where subscriptions to different topic strings have been created in the queue manager network. This enables the queue manager on which a message is published to identify which other queue managers require a copy of the published message, to match their subscriptions.

This approach minimizes the sending of published messages to queue managers on which no matching subscriptions exist. However, the propagation of the subscription knowledge can become a significant overhead, when the number of topic strings being subscribed to is high and constantly changing through frequent subscription creation and deletion.

You can affect performance by adjusting how publications and subscriptions are flowed around your publish/subscribe network. If your network traffic has few publications, and rapid subscription creation, deletion, or change, you can stop subscription information being flowed to all queue managers, and instead forward all publications to all queue managers in the network. You can also restrict the flow of proxy subscriptions and publications for a given topic between connected queue managers, restrict the flow of proxy subscriptions containing wildcards, and reduce the number and transient nature of topic strings.

Individual subscription propagation and *publish everywhere*

Publish everywhere is an alternative to individual subscription propagation. With individual propagation, only publications that have a matching subscription on a queue manager are forwarded to that queue manager. With *publish everywhere*, all publications are forwarded to all queue managers in the network. The receiving queue managers then deliver those publications that match local subscriptions.

Individual subscription propagation

This mechanism results in the least amount of inter-queue manager publication traffic, because only those publications that match subscriptions on a queue manager are sent.

However:

- For each individual topic string that is subscribed to, a proxy subscription is sent to other queue managers in the publish/subscribe topology. The set of queue managers depends on the routing model being used, as described in Planning your distributed publish/subscribe network.
  - This messaging overhead can be significant if there are many thousands of subscriptions to create or delete (for example, recreating all non-durable subscriptions after a restart of a queue manager), or if the set of subscriptions is changing rapidly, and each is to a different topic string.
  - The number of queue managers to which the proxy subscription is propagated also affects the scale of the overhead.
- Proxy subscriptions are flowed to other queue managers using asynchronous messaging. This has the following effect:
  - There is a delay between the creation of a subscription, and the creation, delivery, and processing of the proxy subscription by the other queue managers.
  - Messages that are published at those queue managers in that interval are not delivered to the remote subscription.

Publish everywhere

With this mechanism there is no per topic string proxy subscription overhead on the system. This means that rapid subscription creation, deletion, or change does not result in increased network load and processing.

There is also no delay between creating a subscription and publications being flowed to a queue manager, because all publications are flowed to all queue managers. Therefore there is no window in which publications are not delivered to newly-created remote subscriptions.

However:

- Sending all publications to all queue managers in the publish/subscribe topology, can result in excessive network traffic where publications do not have matching subscriptions on each queue manager.
  - The greater the number of queue managers in the topology, the greater the overhead.

You should consider using the *publish everywhere* mechanism when you expect a publication to be subscribed to from a significant proportion of your queue managers, or where the proxy subscription overheads are too great because of the frequency of subscription changes. You should use individual proxy subscription forwarding in cases where you experience increased messaging traffic when publications are sent to all queue managers, rather than to the queue managers with matching subscriptions.

You can set *publish everywhere* behavior at any level within the topic tree. To enable *publish everywhere*, you set the **PROXYSUB** parameter to FORCE for a high-level topic object. This results in a single wildcard proxy subscription that matches all topics below this topic object in the topic tree. When set it on a clustered topic object, the **PROXYSUB(FORCE)** attribute is propagated to every queue manager in the network, not just the queue manager that the topic was defined on.

Note: When used in a hierarchy, you set **PROXYSUB(FORCE)** individually on each queue manager, so the topology mechanism naturally limits the number of channels. However, when used in a cluster, many additional channels might be started:

- In a topic host routed cluster, channels are started from each queue manager to each topic host queue manager.
- In a direct routed cluster, channels are started from every queue manager to every other queue manager.

The overhead of starting many channels is most pronounced in a direct routed cluster, and can cause performance issues. See "Direct routed publish/subscribe cluster performance" on page 1263.

Other ways of restricting the flow of proxy subscriptions and publications between connected queue managers

Consolidate topic strings
The use of many distinct, transient, topic strings introduces some level of management overhead on each queue manager in the system where publishers or subscriptions are attached. You should periodically assess the use of topic strings to see whether they can be consolidated. Reducing the number and transient nature of topic strings, and therefore the publishers and subscriptions to them, reduces the impact on the system.

Restrict publication and subscription scope
For a given topic, you can use the Publication scope and Subscription scope settings to keep publications and subscriptions local to the queue manager on which they are defined.

Block subscriptions made to wildcarded topics
You can restrict the flow of proxy subscriptions containing wildcards by setting the **Topic** attribute WILDCARD to BLOCK. See Wildcards in proxy subscriptions.

See also "Balancing producers and consumers in publish/subscribe networks" on page 1267

Monitoring proxy subscription traffic in clusters

When considering the load on the system from the proxy subscription traffic, in addition to monitoring the queues listed in"Balancing producers and consumers in publish/subscribe networks" on page 1267, monitor the following cluster queues:

- The SYSTEM.INTER.QMGR.FANREQ queue on the subscriber queue manager.
- The SYSTEM.INTER.QMGR.CONTROL queue on all other queue managers in the cluster.

Any significant message backlog on these queues implies that either the rate of subscription change is too great for the system, or a queue manager is not correctly functioning in the cluster. If you suspect the problem lies with a specific queue manager, check that publish/subscribe support is not disabled for that queue manager. See **PSMODE** in ALTER QMGR.

Related information:

Proxy subscriptions in a publish/subscribe network

Reducing the number of unwanted topics in the topic tree

The performance of a publish/subscribe system is improved by reducing the number of unwanted topics in the topic tree. What is an unwanted topic and how do you remove them?

You can create large numbers of topics without affecting performance adversely. However, some ways of using publish/subscribe result in continually expanding topic trees. An exceptionally large number of topics are created once and never used again. The growing number of topics might become a performance problem.

How can you avoid designs that lead to a large and growing number of unwanted topics? What can you do to help the queue manager remove unwanted topics from the topic tree?

The queue manager recognizes an unwanted topic because it has been unused for 30 minutes. The queue manager removes unused topics from the topic tree for you. The 30 minute duration can be changed by altering the queue manager attribute, **TREELIFE**. You can help the queue manager to remove unwanted topics by making sure that the topic appears to the queue manager to be unused. The section, "What is an unused topic?" explains what an unused topic is.

A programmer, designing any application, and especially designing a long running application, considers its resource usage: how much resource the program requires, are there any unbounded demands, and any resource leaks? Topics are a resource that publish/subscribe programs use. Scrutinize the use of topics just like any other resource a program uses.

What is an unused topic?

Before defining what an unused topic is, what exactly counts as a topic?

When a topic string, such as `USA/Alabama/Auburn`, is converted into a topic, the topic is added to the topic tree. Additional topic nodes, and their corresponding topics, are created in the tree, if necessary. The topic string `USA/Alabama/Auburn` is converted into a tree with three topics.
- `USA`
- `USA/Alabama`
- `USA/Alabama/Auburn`

To display all the topics in the topic tree, use the **runmqsc** command `DISPLAY TPSTATUS('#') TYPE(TOPIC)`.

An unused topic in the topic tree has the following properties.

It is not associated with a topic object

An administrative topic object has a topic string that associates it with a topic. When you define the topic object `Alabama`, if the topic, `USA/Alabama`, it is to be associated with does not exist, the topic is created from the topic string. If the topic does exist, the topic object and the topic are associated together using the topic string.

It does not have a retained publication

A topic with a retained publication results from a publisher putting a message to a topic with the option `MQPMO_RETAIN`.

Use the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama') RETAINED` to check if `USA/Alabama` has a retained publication. The response is `YES` or `NO`.

Use the **runmqsc** command `CLEAR TOPICSTR('USA/Alabama') CLTRTYPE(RETAINED)` to remove a retained publication from `USA/Alabama`.

It has no child topics

USA/Alabama/Auburn is a topic with no child topics. USA/Alabama/Auburn is the direct child topic of USA/Alabama.

Display the direct children of USA/Alabama with the **runmqsc** command DISPLAY TPSTATUS('USA/Alabama/+').

There are no active publishers to the node

An active publisher to a node is an application that has the topic open for output.

For example, an application opens the topic object named **Alabama** with open options MQOO_OUTPUT.

To display active publishers to USA/Alabama and all its children, use the **runmqsc** command DISPLAY TPSTATUS('USA/Alabama/#') TYPE(PUB) ACTCONN.

There are no active subscribers to the node

An active subscriber can either be a durable subscription, or an application that has registered a subscription to a topic with MQSUB, and not closed it.

To display active subscriptions to USA/Alabama, use the **runmqsc** command DISPLAY TPSTATUS('USA/Alabama') TYPE(SUB) ACTCONN.

To display active subscriptions to USA/Alabama and all its children, use the **runmqsc** command DISPLAY TPSTATUS('USA/Alabama/#') TYPE(SUB) ACTCONN.

Managing the number of topics in a topic tree

In summary, there are a number of ways to manage the number of topics in a topic tree.

Display TPCOUNT

Use the **runmqsc** command DISPLAY PUBSUB ALL periodically to display the **TPCOUNT** property. This is the number of topic nodes in the topic tree. If the number is growing it might indicate that a shorter TREELIFE is required, or that a redesign of the topics themselves is required.

Modify TREELIFE

An unused topic has a lifetime of 30 minutes by default. You can make the lifetime of an unused topic smaller.

For example, The **runmqsc** command, ALTER QMGR TREELIFE(900), reduces lifetime of an unused topic from 30 minutes to 15 minutes.

Exceptionally, restart the queue manager

When the queue manager is restarted, the topic tree is reinitialized from topic objects, nodes with retained publications, and durable subscriptions. Topics that had been created by the operation of publisher and subscriber programs are eliminated.

As a last resort, if the growth in unwanted topics has been the cause of performance problems in the past, restart the queue manager.

Related information:
Topic trees

Troubleshooting and support

If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

For an introduction to troubleshooting and support, see "Troubleshooting overview" on page 1276.

There are some initial checks that you can make for your platform to help determine the causes of some common problems. See the appropriate topic for your platform:

- Windows UNIX Linux "Making initial checks on Windows, UNIX and Linux systems" on page 1277
- IBM i "Making initial checks on IBM i" on page 1288
- z/OS "Making initial checks on z/OS" on page 1297

For information about solving problems, see "Making initial checks" on page 1277.

For information about solving problems for IBM MQ Telemetry, see "IBM MQ Telemetry troubleshooting" on page 1828.

For information about solving problems when you are using channel authentication records, see "Channel authentication records troubleshooting" on page 1779.

Information that is produced by IBM MQ can help you to find and resolve problems. For more information, see the following topics:

- "Using logs" on page 1678
- "Using trace" on page 1686
- z/OS "Problem determination on z/OS" on page 1724
- "First Failure Support Technology (FFST" on page 1666

For information about recovering after a problem, see "Recovering after failure" on page 1845.

You can also read the general troubleshooting guidance in the following topics:

- "IBM Support Assistant (ISA)" on page 1635
- "Searching knowledge bases" on page 1638
- "Contacting IBM Software Support" on page 1658
- "Getting product fixes" on page 1665

If an IBM MQ component or command has returned an error, and you want further information about a message written to the screen or the log, you can browse for details of the message, see "Reason codes and exceptions" on page 1314.

Related information:
Troubleshooting and support reference

Troubleshooting overview

Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

A basic troubleshooting strategy at a high level involves:
1. “Recording the symptoms of the problem”
2. “Re-creating the problem”
3. “Eliminating possible causes” on page 1277

Recording the symptoms of the problem

Depending on the type of problem that you have, whether it be with your application, your server, or your tools, you might receive a message that indicates that something is wrong. Always record the error message that you see. As simple as this sounds, error messages sometimes contain codes that might make more sense as you investigate your problem further. You might also receive multiple error messages that look similar but have subtle differences. By recording the details of each one, you can learn more about where your problem exists.

Sources of error messages:
- Problems view
- Local error log
- Eclipse log
- User trace
- Service trace
- Error dialog boxes

Re-creating the problem

Think back to what steps you were doing that led to the problem. Try those steps again to see if you can easily re-create the problem. If you have a consistently repeatable test case, it is easier to determine what solutions are necessary.
- How did you first notice the problem?
- Did you do anything different that made you notice the problem?
- Is the process that is causing the problem a new procedure, or has it worked successfully before?
- If this process worked before, what has changed? (The change can refer to any type of change that is made to the system, ranging from adding new hardware or software, to reconfiguring existing software.)
- What was the first symptom of the problem that you witnessed? Were there other symptoms occurring around the same time?
- Does the same problem occur elsewhere? Is only one machine experiencing the problem or are multiple machines experiencing the same problem?
- What messages are being generated that might indicate what the problem is?

Windows UNIX Linux You can find more information about these types of question in “Making initial checks on Windows, UNIX and Linux systems” on page 1277.

Eliminating possible causes

Narrow the scope of your problem by eliminating components that are not causing the problem. By using a process of elimination, you can simplify your problem and avoid wasting time in areas that are not responsible. Consult the information in this product and other available resources to help you with your elimination process.

- Has anyone else experienced this problem? See: "Searching knowledge bases" on page 1638.
- Is there a fix you can download? See: "Getting product fixes" on page 1665.

Making initial checks

There are some initial checks that you can make that may provide answers to common problems that you may have.

Carry out the initial checks for your platform:

- Windows UNIX Linux "Making initial checks on Windows, UNIX and Linux systems"
- z/OS "Making initial checks on z/OS" on page 1297
- IBM i "Making initial checks on IBM i" on page 1288

For logging and tracing, see the following information:

- "Using logs" on page 1678
- "Using trace" on page 1686

Use the information and general advice given in the subtopics to help you rectify the problem.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

Related information:

Troubleshooting and support reference

Making initial checks on Windows, UNIX and Linux systems

Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

The cause of your problem could be in:

- IBM MQ
- The network
- The application
- Other applications that you have configured to work with IBM MQ

This section contains a list of questions to consider. As you go through the list, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause straight away, they might be useful later if you have to carry out a systematic problem determination exercise.

- "Has IBM MQ run successfully before?" on page 1279
- "Have any changes been made since the last successful run?" on page 1280
- "Are there any error messages or return codes to explain the problem?" on page 1280
- "Can you reproduce the problem?" on page 1281

- “Are you receiving an error code when creating or starting a queue manager? (Windows only)” on page 1281
- “Does the problem affect only remote queues?” on page 1282
- “Have you obtained incorrect output?” on page 1282
- “Are some of your queues failing?” on page 1284
- “Have you failed to receive a response from a PCF command?” on page 1285
- “Has the application run successfully before?” on page 1286
- “Is your application or system running slowly?” on page 1287
- “Does the problem affect specific parts of the network?” on page 1287
- “Does the problem occur at specific times of the day?” on page 1288
- “Is the problem intermittent?” on page 1288

See the following sections for some additional tips for problem determination for system administrators and application developers.

Tips for system administrators

- Check the error logs for messages for your operating system:
  - Windows UNIX Linux “Error logs on Windows, UNIX and Linux systems” on page 1678
  - IBM i “Error logs on IBM i” on page 1683
  - z/OS “Diagnostic information produced on IBM MQ for z/OS” on page 1731
- Check the contents of `qm.ini` for any configuration changes or errors. For more information on changing configuration information, see:
  - Windows UNIX Linux Changing configuration information on Windows, UNIX and Linux systems
  - IBM i Changing configuration information on IBM i
  - z/OS Customizing your queue managers on z/OS
- If your application development teams are reporting something unexpected, you use trace to investigate the problems. For information about using trace, see “Using trace” on page 1686.

Tips for application developers

- Check the return codes from the MQI calls in your applications. For a list of reason codes, see “API reason codes” on page 1315. Use the information provided in the return code to determine the cause of the problem. Follow the steps in the Programmer response sections of the reason code to resolve the problem.
- If you are unsure whether your application is working as expected, for example, you are not unsure of the parameters being passed into the MQI or out of the MQI, you can use trace to collect information about all the inputs and outputs of your MQI calls. For more information about using trace, see “Using trace” on page 1686.
- For more information about handling errors in MQI applications, see Handling program errors.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

z/OS "Making initial checks on z/OS" on page 1297
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

IBM i "Making initial checks on IBM i" on page 1288
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Making initial checks" on page 1277
There are some initial checks that you can make that may provide answers to common problems that you may have.

"IBM Support Assistant (ISA)" on page 1635
The IBM Support Assistant (ISA) helps you to resolve questions and problems with IBM software products by providing access to support-related information and troubleshooting tools.

"Reason codes and exceptions" on page 1314
You can use the following messages and reason codes to help you solve problems with your IBM MQ components or applications.

Related tasks:

"Searching knowledge bases" on page 1638
If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Related reference:

"PCF reason codes" on page 1541
Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

Related information:

Troubleshooting and support reference

Has IBM MQ run successfully before?

If IBM MQ has not run successfully before, it is likely that you have not yet set it up correctly. See Installing IBM MQ and select the platform, or platforms, that your enterprise uses to check that you have installed the product correctly.

To run the verification procedure, see:

- Verifying a server installation
- Verifying a client installation

Also look at Configuring for information about post-installation configuration of IBM MQ.

Have any changes been made since the last successful run?

Changes that have been made to your IBM MQ configuration, maintenance updates, or changes to other programs that interact with IBM MQ could be the cause of your problem.

When you are considering changes that might recently have been made, think about the IBM MQ system, and also about the other programs it interfaces with, the hardware, and any new applications. Consider also the possibility that a new application that you are not aware of might have been run on the system.

- Have you changed, added, or deleted any queue definitions?
- Have you changed or added any channel definitions? Changes might have been made to either IBM MQ channel definitions or any underlying communications definitions required by your application.
- Do your applications deal with return codes that they might get as a result of any changes you have made?
- Have you changed any component of the operating system that could affect the operation of IBM MQ? For example, have you modified the Windows Registry.

Have you applied any maintenance updates?

If you have applied a maintenance update to IBM MQ, check that the update action completed successfully and that no error message was produced.

- Did the update have any special instructions?
- Was any test run to verify that the update was applied correctly and completely?
- Does the problem still exist if IBM MQ is restored to the previous maintenance level?
- If the installation was successful, check with the IBM Support Center for any maintenance package errors.
- If a maintenance package has been applied to any other program, consider the effect it might have on the way IBM MQ interfaces with it.

Are there any error messages or return codes to explain the problem?

You might find error messages or return codes that help you to determine the location and cause of your problem.

IBM MQ uses error logs to capture messages concerning its own operation, any queue managers that you start, and error data coming from the channels that are in use. Check the error logs to see if any messages have been recorded that are associated with your problem.

IBM MQ also logs errors in the Windows Application Event Log. On Windows, check if the Windows Application Event Log shows any IBM MQ errors. To open the log, from the Computer Management panel, expand **Event Viewer** and select **Application**.

Windows UNIX Linux For information about the locations and contents of the error logs, see "Error logs on Windows, UNIX and Linux systems" on page 1678

For each IBM MQ Message Queue Interface (MQI) and IBM MQ Administration Interface (MQAI) call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call. If your application gets a return code indicating that a Message Queue Interface (MQI) call has failed, check the reason code to find out more about the problem.

For a list of reason codes, see "API completion and reason codes" on page 1314.

Detailed information on return codes is contained within the description of each MQI call.

Related reference:

"PCF reason codes" on page 1541
Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

"Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes" on page 1622
IBM MQ can use Secure Sockets Layer (SSL) with the various communication protocols. Use this topic to identify the error codes that can be returned by SSL.

"WCF custom channel exceptions" on page 1627
Diagnostic messages are listed in this topic in numeric order, grouped according to the part of the WCF custom channel from which they originate.

Related information:

IBM MQ AMQ messages

z/OS IBM MQ for z/OS messages, completion, and reason codes

Troubleshooting and support reference

Can you reproduce the problem?

If you can reproduce the problem, consider the conditions under which it is reproduced:

- Is it caused by a command or an equivalent administration request?

  Does the operation work if it is entered by another method? If the command works if it is entered on the command line, but not otherwise, check that the command server has not stopped, and that the queue definition of the SYSTEM.ADMIN.COMMAND.QUEUE has not been changed.
- Is it caused by a program? Does it fail on all IBM MQ systems and all queue managers, or only on some?
- Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application to see if it is in error.

Are you receiving an error code when creating or starting a queue manager? (Windows only)

If the IBM MQ Explorer, or the **amqmdain** command, fails to create or start a queue manager, indicating an authority problem, it might be because the user under which the IBM MQ Windows service is running has insufficient rights.

Ensure that the user with which the IBM MQ Windows service is configured has the rights described in User rights required for an IBM MQ Windows Service. By default this service is configured to run as the **MUSR_MQADMIN** user. For subsequent installations, the Prepare IBM MQ Wizard creates a user account named **MUSR_MQADMINx** , where x is the next available number representing a user ID that does not exist.

Does the problem affect only remote queues?

Things to check if the problem affects only remote queues.

If the problem affects only remote queues, perform the following checks:

- Check that required channels have started, can be triggered, and any required initiators are running.
- Check that the programs that should be putting messages to the remote queues have not reported problems.
- If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.
- Check the error logs for messages indicating channel errors or problems.
- If necessary, start the channel manually.

Have you obtained incorrect output?

In this section, *incorrect output* refers to your application: not receiving a message that you were expecting it to receive; receiving a message containing unexpected or corrupted information; receiving a message that you were not expecting it to receive, for example, one that was destined for a different application.

Messages that do not arrive on the queue

If messages do not arrive when you are expecting them, check for the following:

- Has the message been put on the queue successfully?
  - Has the queue been defined correctly? For example, is MAXMSGL sufficiently large?
  - Is the queue enabled for putting?
  - Is the queue already full?
  - Has another application got exclusive access to the queue?
- Are you able to get any messages from the queue?
  - Do you need to take a sync point?

    If messages are being put or retrieved within sync point, they are not available to other tasks until the unit of recovery has been committed.
  - Is your wait interval long enough?

    You can set the wait interval as an option for the MQGET call. Ensure that you are waiting long enough for a response.
  - Are you waiting for a specific message that is identified by a message or correlation identifier (`MsgId` or `CorrelId`)?

    Check that you are waiting for a message with the correct `MsgId` or `CorrelId`. A successful MQGET call sets both these values to that of the message retrieved, so you might need to reset these values in order to get another message successfully.

    Also, check whether you can get other messages from the queue.
  - Can other applications get messages from the queue?
  - Was the message you are expecting defined as persistent?

    If not, and IBM MQ has been restarted, the message has been lost.
  - Has another application got exclusive access to the queue?

If you cannot find anything wrong with the queue, and IBM MQ is running, check the process that you expected to put the message onto the queue for the following:

- Did the application start?

  If it should have been triggered, check that the correct trigger options were specified.
- Did the application stop?
- Is a trigger monitor running?

- Was the trigger process defined correctly?
- Did the application complete correctly?

  Look for evidence of an abnormal end in the job log.
- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they can conflict with one another. For example, suppose one transaction issues an MQGET call with a buffer length of zero to find out the length of the message, and then issues a specific MQGET call specifying the `MsgId` of that message. However, in the meantime, another transaction issues a successful MQGET call for that message, so the first application receives a reason code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multiple server environment must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If so, refer to the subsequent information in this topic.

Messages that contain unexpected or corrupted information

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following:
- Has your application, or the application that put the message onto the queue, changed?

  Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

  For example, the format of the message data might have been changed, in which case, both applications must be recompiled to pick up the changes. If one application has not been recompiled, the data will appear corrupted to the other.
- Is an application sending messages to the wrong queue?

  Check that the messages your application is receiving are not intended for an application servicing a different queue. If necessary, change your security definitions to prevent unauthorized applications from putting messages on to the wrong queues.

  If your application uses an alias queue, check that the alias points to the correct queue.
- Has the trigger information been specified correctly for this queue?

  Check that your application should have started; or should a different application have started?

If these checks do not enable you to solve the problem, check your application logic, both for the program sending the message, and for the program receiving it.

Problems with incorrect output when using distributed queues

If your application uses distributed queues, consider the following points:
- Has IBM MQ been correctly installed on both the sending and receiving systems, and correctly configured for distributed queuing?
- Are the links available between the two systems?

  Check that both systems are available, and connected to IBM MQ. Check that the connection between the two systems is active.

  You can use the MQSC command PING against either the queue manager (PING QMGR) or the channel (PING CHANNEL) to verify that the link is operable.
- Is triggering set on in the sending system?
- Is the message for which you are waiting a reply message from a remote system?

  Check that triggering is activated in the remote system.
- Is the queue already full?

If so, check if the message has been put onto the dead-letter queue.

The dead-letter queue header contains a reason or feedback code explaining why the message could not be put onto the target queue. See Using the dead-letter (undelivered message) queue and MQDLH - Dead-letter header for information about the dead-letter queue header structure.

- Is there a mismatch between the sending and receiving queue managers?

  For example, the message length could be longer than the receiving queue manager can handle.
- Are the channel definitions of the sending and receiving channels compatible?

  For example, a mismatch in sequence number wrap can stop the distributed queuing component. See Distributed queuing and clusters for more information about distributed queuing.
- Is data conversion involved? If the data formats between the sending and receiving applications differ, data conversion is necessary. Automatic conversion occurs when the MQGET call is issued if the format is recognized as one of the built-in formats.

  If the data format is not recognized for conversion, the data conversion exit is taken to allow you to perform the translation with your own routines.

  Refer to Data conversion for further information about data conversion.

Are some of your queues failing?

If you suspect that the problem occurs with only a subset of queues, check the local queues that you think are having problems.

Perform the following checks:

1. Display the information about each queue. You can use the MQSC command DISPLAY QUEUE to display the information.
2. Use the data displayed to do the following checks:
   - If CURDEPTH is at MAXDEPTH, the queue is not being processed. Check that all applications are running normally.
   - If CURDEPTH is not at MAXDEPTH, check the following queue attributes to ensure that they are correct:
     - If triggering is being used:
       - Is the trigger monitor running?
       - Is the trigger depth too great? That is, does it generate a trigger event often enough?
       - Is the process name correct?
       - Is the process available and operational?
     - Can the queue be shared? If not, another application could already have it open for input.
     - Is the queue enabled appropriately for GET and PUT?
   - If there are no application processes getting messages from the queue, determine why this is so. It could be because the applications need to be started, a connection has been disrupted, or the MQOPEN call has failed for some reason.

     Check the queue attributes IPPROCS and OPPROCS. These attributes indicate whether the queue has been opened for input and output. If a value is zero, it indicates that no operations of that type can occur. The values might have changed; the queue might have been open but is now closed.

     You need to check the status at the time you expect to put or get a message.

If you are unable to solve the problem, contact your IBM Support Center for help.

Have you failed to receive a response from a PCF command?

Considerations if you have issued a command but have not received a response.

If you have issued a command but have not received a response, consider the following checks:

- Is the command server running?

  Work with the **dspmqcsv** command to check the status of the command server.

  - If the response to this command indicates that the command server is not running, use the **strmqcsv** command to start it.
  - If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.
- Has a reply been sent to the dead-letter queue?

  The dead-letter queue header structure contains a reason or feedback code describing the problem. See MQDLH - Dead-letter header and Using the dead-letter (undelivered message) queue for information about the dead-letter queue header structure (MQDLH).

  If the dead-letter queue contains messages, you can use the provided browse sample application (amqsbcg) to browse the messages using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.
- Has a message been sent to the error log?

  See "Error log directories" on page 1681 for further information.
- Are the queues enabled for put and get operations?
- Is the *WaitInterval* long enough?

  If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See WaitInterval (MQLONG) for information about the *WaitInterval* field, and completion and reason codes from MQGET.)
- If you are using your own application program to put commands onto the SYSTEM.ADMIN.COMMAND.QUEUE, do you need to take a sync point?

  Unless you have excluded your request message from sync point, you need to take a sync point before receiving reply messages.
- Are the MAXDEPTH and MAXMSGL attributes of your queues set sufficiently high?
- Are you using the *CorrelId* and *MsgId* fields correctly?

  Set the values of *MsgId* and *CorrelId* in your application to ensure that you receive all messages from the queue.

Try stopping the command server and then restarting it, responding to any error messages that are produced.

If the system still does not respond, the problem could be with either a queue manager or the whole of the IBM MQ system. First, try stopping individual queue managers to isolate a failing queue manager. If this step does not reveal the problem, try stopping and restarting IBM MQ, responding to any messages that are produced in the error log.

If the problem still occurs after restart, contact your IBM Support Center for help.

Has the application run successfully before?

Use the information in this topic to help diagnose common problems with applications.

If the problem appears to involve one particular application, consider whether the application has run successfully before.

Before you answer **Yes** to this question, consider the following:

- Have any changes been made to the application since it last ran successfully?

  If so, it is likely that the error lies somewhere in the new or modified part of the application. Take a look at the changes and see if you can find an obvious reason for the problem. Is it possible to retry using a back level of the application?
- Have all the functions of the application been fully exercised before?

  Could it be that the problem occurred when part of the application that had never been invoked before was used for the first time? If so, it is likely that the error lies in that part of the application. Try to find out what the application was doing when it failed, and check the source code in that part of the program for errors.

  If a program has been run successfully on many previous occasions, check the current queue status and the files that were being processed when the error occurred. It is possible that they contain some unusual data value that invokes a rarely-used path in the program.
- Does the application check all return codes?

  Has your IBM MQ system been changed, perhaps in a minor way, such that your application does not check the return codes it receives as a result of the change. For example, does your application assume that the queues it accesses can be shared? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?
- Does the application run on other IBM MQ systems?

  Could it be that there is something different about the way that this IBM MQ system is set up that is causing the problem? For example, have the queues been defined with the same message length or priority?

Before you look at the code, and depending upon which programming language the code is written in, examine the output from the translator, or the compiler and linkage editor, to see if any errors have been reported.

If your application fails to translate, compile, or link-edit into the load library, it will also fail to run if you attempt to invoke it. See Developing applications for information about building your application.

If the documentation shows that each of these steps was accomplished without error, consider the coding logic of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See the following section for some examples of common errors that cause problems with IBM MQ applications.

Common programming errors

The errors in the following list illustrate the most common causes of problems encountered while running IBM MQ programs. Consider the possibility that the problem with your IBM MQ system could be caused by one or more of these errors:

- Assuming that queues can be shared, when they are in fact exclusive.
- Passing incorrect parameters in an MQI call.
- Passing insufficient parameters in an MQI call. This might mean that IBM MQ cannot set up completion and reason codes for your application to process.
- Failing to check return codes from MQI requests.
- Passing variables with incorrect lengths specified.

- Passing parameters in the wrong order.
- Failing to initialize `MsgId` and `CorrelId` correctly.
- Failing to initialize `Encoding` and `CodedCharSetId` following MQRC_TRUNCATED_MSG_ACCEPTED.

Is your application or system running slowly?

If your application is running slowly, it might be in a loop, or waiting for a resource that is not available, or there might be a performance problem.

Perhaps your system is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to occur at some other time.)

A performance problem might be caused by a limitation of your hardware.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly-designed application program is probably to blame. This could appear to be a problem that only occurs when certain queues are accessed.

If the performance issue persists, the problem might lie with IBM MQ itself. If you suspect this, contact your IBM Support Center for help.

A common cause of slow application performance, or the build up of messages on a queue (usually a transmission queue) is one or more applications that write persistent messages outside a unit of work; for more information, see Message persistence.

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check that the connection between the two systems is available, and that the intercommunication component of IBM MQ has started.

Check that messages are reaching the transmission queue, and check the local queue definition of the transmission queue and any remote queues.

Have you made any network-related changes, or changed any IBM MQ definitions, that might account for the problem?

Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it could be that it depends on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, so these are the times when load-dependent problems are most likely to occur. (If your IBM MQ network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

Is the problem intermittent?

An intermittent problem could be caused by the way that processes can run independently of each other. For example, a program might issue an MQGET call without specifying a wait option before an earlier process has completed. An intermittent problem might also be seen if your application tries to get a message from a queue before the call that put the message has been committed.

Making initial checks on IBM i

IBM i

Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

The cause of your problem could be in any of the following:
- Hardware
- Operating system
- Related software, for example, a language compiler
- The network
- The IBM MQ product
- Your IBM MQ application
- Other applications
- Site operating procedures

This section contains a list of questions to consider. As you go through the list, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause straight away, they might be useful later if you have to carry out a systematic problem determination exercise.

The following steps are intended to help you isolate the problem and are taken from the viewpoint of an IBM MQ application. Check all the suggestions at each stage.

1. Has IBM MQ for IBM i run successfully before?

   Yes Proceed to Step 2.

   No It is likely that you have not installed or set up IBM MQ correctly.
2. Has the IBM MQ application run successfully before?

   Yes Proceed to Step 3 on page 1289.

   No Consider the following:
   a. The application might have failed to compile or link, and fails if you attempt to invoke it. Check the output from the compiler or linker.

      Refer to the appropriate programming language reference information, or see Developing applications, for information about how to build your application.
   b. Consider the logic of the application. For example, do the symptoms of the problem indicate that a function is failing and, therefore, that a piece of code is in error.

      Check the following common programming errors:

- Assuming that queues can be shared, when they are in fact exclusive.
- Trying to access queues and data without the correct security authorization.
- Passing incorrect parameters in an MQI call; if the wrong number of parameters is passed, no attempt can be made to complete the completion code and reason code fields, and the task is ended abnormally.
- Failing to check return codes from MQI requests.
- Using incorrect addresses.
- Passing variables with incorrect lengths specified.
- Passing parameters in the wrong order.
- Failing to initialize `MsgId` and `CorrelId` correctly.

3. Has the IBM MQ application changed since the last successful run?

   Yes It is likely that the error lies in the new or modified part of the application. Check all the changes and see if you can find an obvious reason for the problem.

   a. Have all the functions of the application been fully exercised before?

      Could it be that the problem occurred when part of the application that had never been invoked before was used for the first time? If so, it is likely that the error lies in that part of the application. Try to find out what the application was doing when it failed, and check the source code in that part of the program for errors.

   b. If the program has run successfully before, check the current queue status and files that were being processed when the error occurred. It is possible that they contain some unusual data value that causes a rarely used path in the program to be invoked.

   c. The application received an unexpected MQI return code. For example:

      - Does your application assume that the queues it accesses are shareable? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?
      - Have any queue definition or security profiles been changed? An MQOPEN call could fail because of a security violation; can your application recover from the resulting return code?

      See MQI Applications reference for your programming language for a description of each return code.

   d. If you have applied any PTF to IBM MQ for IBM i, check that you received no error messages when you installed the PTF.

   No Ensure that you have eliminated all the preceding suggestions and proceed to Step 4.

4. Has the server system remained unchanged since the last successful run?

   Yes Proceed to “Problem characteristics” on page 1291.

   No Consider all aspects of the system and review the appropriate documentation on how the change might have affected the IBM MQ application. For example :

   - Interfaces with other applications
   - Installation of new operating system or hardware
   - Application of PTFs
   - Changes in operating procedures

See the following sections for some additional tips for problem determination for system administrators and application developers.

Tips for system administrators

- Check the error logs for messages for your operating system:
  - Windows UNIX Linux “Error logs on Windows, UNIX and Linux systems” on page 1678

- IBM i "Error logs on IBM i" on page 1683
- z/OS "Diagnostic information produced on IBM MQ for z/OS" on page 1731
- Check the contents of `qm.ini` for any configuration changes or errors. For more information on changing configuration information, see:
  - Windows UNIX Linux Changing configuration information on Windows, UNIX and Linux systems
  - IBM i Changing configuration information on IBM i
  - z/OS Customizing your queue managers on z/OS
- If your application development teams are reporting something unexpected, you use trace to investigate the problems. For information about using trace, see "Using trace" on page 1686.

Tips for application developers

- Check the return codes from the MQI calls in your applications. For a list of reason codes, see "API reason codes" on page 1315. Use the information provided in the return code to determine the cause of the problem. Follow the steps in the Programmer response sections of the reason code to resolve the problem.
- If you are unsure whether your application is working as expected, for example, you are not unsure of the parameters being passed into the MQI or out of the MQI, you can use trace to collect information about all the inputs and outputs of your MQI calls. For more information about using trace, see "Using trace" on page 1686.
- For more information about handling errors in MQI applications, see Handling program errors.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Required authority" on page 1293
Some IBM MQ commands rely on using IBM i system commands for creating and managing objects, files, and libraries.

"Making initial checks on Windows, UNIX and Linux systems" on page 1277
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Making initial checks on z/OS" on page 1297
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Making initial checks" on page 1277
There are some initial checks that you can make that may provide answers to common problems that you may have.

"IBM Support Assistant (ISA)" on page 1635
The IBM Support Assistant (ISA) helps you to resolve questions and problems with IBM software products by providing access to support-related information and troubleshooting tools.

"Reason codes and exceptions" on page 1314
You can use the following messages and reason codes to help you solve problems with your IBM MQ components or applications.

Related tasks:

"Searching knowledge bases" on page 1638
If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Related reference:

"Determining problems with IBM MQ for IBM i applications" on page 1295
If you have not yet found the cause, start to look at the problem in greater detail.

"PCF reason codes" on page 1541
Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

Related information:

Troubleshooting and support reference

Problem characteristics

Perhaps by using the preliminary checks, you have found the cause of the problem. If so, you can probably now resolve it, possibly with the help of other sections in the IBM MQ product documentation, and in the libraries of other licensed programs.

If you have not yet found the cause, start to look at the problem in greater detail. Use the following questions as pointers to the problem. Answering the appropriate question, or questions, should lead you to the cause of the problem:

- "Can you reproduce the problem?"
- "Is the problem intermittent?"
- "Problems with commands" on page 1292
- "Does the problem affect all users of the IBM MQ for IBM i application?" on page 1292
- "Does the problem affect specific parts of the network?" on page 1292
- "Does the problem occur only on IBM MQ?" on page 1292
- "Does the problem occur at specific times of the day?" on page 1292
- "Have you failed to receive a response from a command?" on page 1293

Can you reproduce the problem?

If you can reproduce the problem, consider the conditions under which you do so:

- Is it caused by a command?

  Does the operation work if it is entered by another method? If the command works if it is entered on the command line, but not otherwise, check that the command server has not stopped. You must also check that the queue definition of the `SYSTEM.ADMIN.COMMAND.QUEUE` has not been changed.
- Is it caused by a program? If so, does it fail in batch? Does it fail on all IBM MQ for IBM i systems, or only on some?
- Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application to see if it is in error.
- Does the problem occur with any queue manager, or when connected to one specific queue manager?
- Does the problem occur with the same type of object on any queue manager, or only one particular object? What happens after this object has been cleared or redefined?
- Is the problem independent of any message persistence settings?
- Does the problem occur only when sync points are used?
- Does the problem occur only when one or more queue-manager events are enabled?

Is the problem intermittent?

An intermittent problem might be caused by failing to take into account the fact that processes can run independently of each other. For example, a program might issue an MQGET call, without specifying a wait option, before an earlier process has completed. You might also encounter this problem if your application tries to get a message from a queue while the call that put the message is in-doubt (that is, before it has been committed or backed out).

Problems with commands

Use this information to avoid potential problems with special characters. Be careful when including special characters, for example backslash (\) and quotation marks (") characters, in descriptive text for some commands. If you use either of these characters in descriptive text, precede them with a backslash (\) character, for example:

- Enter \\ if you need a backslash (\) character in your text.
- Enter \" if you need quotation marks (") characters in your text.

Queue managers and their associated object names are case-sensitive. By default, IBM i uses uppercase characters, unless you surround the name in apostrophe (') characters.

For example, `MYQUEUE` and `myqueue` translate to `MYQUEUE`, whereas `'myqueue'` translates to `myqueue`.

Does the problem affect all users of the IBM MQ for IBM i application?

If the problem affects only some users, look for differences in how the users configure their systems and queue manager settings.

Check the library lists and user profiles. Can the problem be circumvented by having `*ALLOBJ` authority?

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check these points:

- Is the connection between the two systems available, and has the intercommunication component of IBM MQ for IBM i started?

  Check that messages are reaching the transmission queue, the local queue definition of the transmission queue, and any remote queues.
- Have you made any network-related changes that might account for the problem or changed any IBM MQ for IBM i definitions?
- Can you distinguish between a channel definition problem and a channel message problem?

  For example, redefine the channel to use an empty transmission queue. If the channel starts correctly, the definition is correctly configured.

Does the problem occur only on IBM MQ?

If the problem occurs only on this version of IBM MQ, check the appropriate database on RETAIN, or the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, to ensure that you have applied all the relevant PTFs.

Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it might be that it is dependent on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, and so these times are when load-dependent problems are most likely to occur. (If your IBM MQ for IBM i network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

Have you failed to receive a response from a command?

If you have issued a command but you have not received a response, consider the following questions:

- Is the command server running?

  Work with the **DSPMQMCSVR** command to check the status of the command server.

  – If the response to this command indicates that the command server is not running, use the **STRMQMCSVR** command to start it.

  – If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.
- Has a reply been sent to the dead-letter queue?

  The dead-letter queue header structure contains a reason or feedback code describing the problem. See MQDLH - Dead-letter header for information about the dead-letter queue header structure (MQDLH).

  If the dead-letter queue contains messages, you can use the provided browse sample application (amqsbcg) to browse the messages using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.
- Has a message been sent to the error log?

  See "Error logs on IBM i" on page 1683 for further information.
- Are the queues enabled for put and get operations?
- Is the `WaitInterval` long enough?

  If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See Getting messages from a queue using the MQGET call for more information about the `WaitInterval` field, and completion and reason codes from MQGET.)
- If you are using your own application program to put commands onto the SYSTEM.ADMIN.COMMAND.QUEUE, do you need to take a sync point?

  Unless you have excluded your request message from sync point, you must take a sync point before attempting to receive reply messages.
- Are the MAXDEPTH and MAXMSGL attributes of your queues set sufficiently high?
- Are you using the `CorrelId` and `MsgId` fields correctly?

  Set the values of `MsgId` and `CorrelId` in your application to ensure that you receive all messages from the queue.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Required authority"
Some IBM MQ commands rely on using IBM i system commands for creating and managing objects, files, and libraries.

Related reference:

"Determining problems with IBM MQ for IBM i applications" on page 1295
If you have not yet found the cause, start to look at the problem in greater detail.

Required authority

Some IBM MQ commands rely on using IBM i system commands for creating and managing objects, files, and libraries.

For example, `CRTMQM` (create queue manager) and `DLTMQM` (delete queue manager). Similarly some IBM MQ program code, for example a queue manager, relies on using IBM i system programs.

To enable this reliance, the commands and programs listed must either have `*PUBLIC *USE` authority, or explicit `*USE` authority to the IBM MQ user profiles `QMQM` and `QMQMADM`.

Such authority is applied automatically as part of the install process, and you do not need to apply it yourself.

However, if you encounter problems, here is how to do it manually:

1. Set the authorities for commands using GRTOBJAUT with an OBJTYPE(*CMD) parameter, for example:

   ```
   GRTOBJAUT OBJ(QSYS/ADDLIBLE) OBJTYPE(*CMD) USER(QMQMADM) AUT(*USE)
   ```

   - QSYS/ADDLIBLE
   - QSYS/ADDPFM
   - QSYS/CALL
   - QSYS/CHGCURLIB
   - QSYS/CHGJOB
   - QSYS/CRTJRN
   - QSYS/CRTJRNRCV
   - QSYS/CRTJOBQ
   - QSYS/CRTJOBD
   - QSYS/CRTLIB
   - QSYS/CRTMSGQ
   - QSYS/CRTPF
   - QSYS/CRTPGM
   - QSYS/CRTSRCPF
   - QSYS/DLTJRN
   - QSYS/DLTJRNRCV
   - QSYS/DLTLIB
   - QSYS/DLTMSGQ
   - QSYS/OVRPRTF
   - QSYS/RCLACTGRP
   - QSYS/RTVJRNE
   - QSYS/RCVJRNE
   - QSYS/SBMJOB
2. Set the authorities for programs using GRTOBJAUT with an OBJTYPE(*PGM) parameter, for example:

   ```
   GRTOBJAUT OBJ(QSYS/QWTSETP) OBJTYPE(*PGM) USER(QMQMADM) AUT(*USE)
   ```

   - QSYS/QWTSETP(*PGM)
   - QSYS/QSYRLSPH(*PGM)
   - QSYS/QSYGETPH(*PGM)

Determining problems with IBM MQ for IBM i applications

IBM i

If you have not yet found the cause, start to look at the problem in greater detail.

This section contains information about problems you might encounter with IBM MQ applications, commands, and messages. Use the following questions as pointers to the problem. Answering the appropriate question, or questions, should lead you to the cause of the problem.

Are some of your queues working?

If you suspect that the problem occurs with only a subset of queues, select the name of a local queue that you think is having problems.

1. Display the information about this queue, using WRKMQMQSTS or DSPMQMQ.
2. Use the data displayed to do the following checks:
   - If CURDEPTH is at MAXDEPTH, the queue is not being processed. Check that all applications are running normally.
   - If CURDEPTH is not at MAXDEPTH, check the following queue attributes to ensure that they are correct:
     - If triggering is being used:
       - Is the trigger monitor running?
       - Is the trigger depth too large?
       - Is the process name correct?
     - Can the queue be shared? If not, another application might already have it open for input.
     - Is the queue enabled appropriately for GET and PUT?
   - If there are no application processes getting messages from the queue, determine why (for example, because the applications must be started, a connection has been disrupted, or because the MQOPEN call has failed for some reason).

If you cannot solve the problem, contact your IBM support center for help.

Does the problem affect only remote queues?

If the problem affects only remote queues, check the subsequent points:

1. Check that the programs that should be putting messages to the remote queues have run successfully.
2. If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.
3. If necessary, start the channel manually. See Distributed queuing and clusters.
4. Check the channel with a PING command.

Are messages failing to arrive on the queue?

If messages do not arrive when you are expecting them, check for the following:

- Have you selected the correct queue manager, that is, the default queue manager or a named queue manager?
- Has the message been put on the queue successfully?
  - Has the queue been defined correctly, for example, is MAXMSGLEN sufficiently large?
  - Can applications put messages on the queue (is the queue enabled for putting)?
  - If the queue is already full, it might mean that an application was unable to put the required message on the queue.

- Can you get the message from the queue?
  - Must you take a sync point?

    If messages are being put or retrieved within sync point, they are not available to other tasks until the unit of recovery has been committed.
  - Is your timeout interval long enough?
  - Are you waiting for a specific message that is identified by a message identifier or correlation identifier (*MsgId* or *CorrelId*)?

    Check that you are waiting for a message with the correct *MsgId* or *CorrelId*. A successful MQGET call sets both these values to that of the message retrieved, so you might need to reset these values in order to get another message successfully.

    Also check if you can get other messages from the queue.
  - Can other applications get messages from the queue?
  - Was the message you are expecting defined as persistent?

    If not, and IBM MQ for IBM i has been restarted, the message has been lost.

If you cannot find anything wrong with the queue, and the queue manager itself is running, make the following checks on the process that you expected to put the message on to the queue:

- Did the application start?

  If it should have been triggered, check that the correct trigger options were specified.
- Is a trigger monitor running?
- Was the trigger process defined correctly?
- Did it complete correctly?

  Look for evidence of an abnormal end in the job log.
- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they might occasionally conflict with one another. For example, one transaction might issue an MQGET call with a buffer length of zero to find out the length of the message, and then issue a specific MQGET call specifying the *MsgId* of that message. However, in the meantime, another transaction might have issued a successful MQGET call for that message, so the first application receives a completion code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multi-server environment must be designed to cope with this situation.

Consider that the message might have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If so, see "Are unexpected messages received when using distributed queues?" on page 1297.

Do messages contain unexpected or corrupted information?

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following points:

- Has your application, or the application that put the message on to the queue, changed?

  Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

  For example, a copyfile formatting the message might have been changed, in which case, recompile both applications to pick up the changes. If one application has not been recompiled, the data appears corrupted to the other.
- Is an application sending messages to the wrong queue?

  Check that the messages your application is receiving are not intended for an application servicing a different queue. If necessary, change your security definitions to prevent unauthorized applications from putting messages on to the wrong queues.

If your application has used an alias queue, check that the alias points to the correct queue.

- Has the trigger information been specified correctly for this queue?

  Check that your application should have been started, or should a different application have been started?
- Has the CCSID been set correctly, or is the message format incorrect because of data conversion.

If these checks do not enable you to solve the problem, check your application logic, both for the program sending the message, and for the program receiving it.

Are unexpected messages received when using distributed queues?

If your application uses distributed queues, consider the following points:

- Has distributed queuing been correctly installed on both the sending and receiving systems?
- Are the links available between the two systems?

  Check that both systems are available, and connected to IBM MQ for IBM i. Check that the connection between the two systems is active.
- Is triggering set on in the sending system?
- Is the message you are waiting for a reply message from a remote system?

  Check that triggering is activated in the remote system.
- Is the queue already full?

  If so, it might mean that an application was unable to put the required message on to the queue. Check that the message has been put onto the undelivered-message queue.

  The dead-letter queue message header (dead-letter header structure) contains a reason or feedback code explaining why the message could not be put on to the target queue. See MQDLH - Dead-letter header or IBM i Application Programming Reference (ILE/RPG), as appropriate, for information about the dead-letter header structure.
- Is there a mismatch between the sending and receiving queue managers?

  For example, the message length could be longer than the receiving queue manager can handle.
- Are the channel definitions of the sending and receiving channels compatible?

  For example, a mismatch in sequence number wrap stops the distributed queuing component. See Distributed queuing and clusters.

Making initial checks on z/OS

Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

The cause of your problem could be in:

- IBM MQ
- The network
- The application
- Other applications that you have configured to work with IBM MQ

This section contains a list of questions to consider. As you go through the list, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause straight away, they might be useful later if you have to carry out a systematic problem determination exercise.

- “Has IBM MQ for z/OS run successfully before?” on page 1299
- “Have you applied any APARs or PTFs?” on page 1300
- “Are there any error messages, return codes or other error conditions?” on page 1300

- “Has your application or IBM MQ for z/OS stopped processing work?” on page 1302
- “Is there a problem with the IBM MQ queues?” on page 1302
- “Are some of your queues working?” on page 1303
- “Are the correct queues defined?” on page 1304
- “Does the problem affect only remote or cluster queues?” on page 1304
- “Does the problem affect only shared queues?” on page 1305
- “Does the problem affect specific parts of the network?” on page 1306
- “Problems that occur at specific times of the day or affect specific users” on page 1306
- “Is the problem intermittent or does the problem occur with all z/OS, CICS, or IMS systems?” on page 1306
- “Has the application run successfully before?” on page 1307
- “Have any changes been made since the last successful run?” on page 1308
- “Do you have a program error?” on page 1309
- “Has there been an abend?” on page 1310
- “Have you obtained incorrect output?” on page 1311
- “Can you reproduce the problem?” on page 1311
- “Have you failed to receive a response from an MQSC command?” on page 1312
- “Is your application or IBM MQ for z/OS running slowly?” on page 1313

See the following sections for some additional tips for problem determination for system administrators and application developers.

Tips for system administrators

- Check the error logs for messages for your operating system:
  - Windows UNIX Linux “Error logs on Windows, UNIX and Linux systems” on page 1678
  - IBM i “Error logs on IBM i” on page 1683
  - z/OS “Diagnostic information produced on IBM MQ for z/OS” on page 1731
- Check the contents of `qm.ini` for any configuration changes or errors. For more information on changing configuration information, see:
  - Windows UNIX Linux Changing configuration information on Windows, UNIX and Linux systems
  - IBM i Changing configuration information on IBM i
  - z/OS Customizing your queue managers on z/OS
- If your application development teams are reporting something unexpected, you use trace to investigate the problems. For information about using trace, see “Using trace” on page 1686.

Tips for application developers

- Check the return codes from the MQI calls in your applications. For a list of reason codes, see “API reason codes” on page 1315. Use the information provided in the return code to determine the cause of the problem. Follow the steps in the Programmer response sections of the reason code to resolve the problem.
- If you are unsure whether your application is working as expected, for example, you are not unsure of the parameters being passed into the MQI or out of the MQI, you can use trace to collect information about all the inputs and outputs of your MQI calls. For more information about using trace, see “Using trace” on page 1686.
- For more information about handling errors in MQI applications, see Handling program errors.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Making initial checks on Windows, UNIX and Linux systems" on page 1277
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

IBM i "Making initial checks on IBM i" on page 1288
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Making initial checks" on page 1277
There are some initial checks that you can make that may provide answers to common problems that you may have.

"IBM Support Assistant (ISA)" on page 1635
The IBM Support Assistant (ISA) helps you to resolve questions and problems with IBM software products by providing access to support-related information and troubleshooting tools.

"Reason codes and exceptions" on page 1314
You can use the following messages and reason codes to help you solve problems with your IBM MQ components or applications.

Related tasks:

"Searching knowledge bases" on page 1638
If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Related reference:

"PCF reason codes" on page 1541
Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

Related information:

Troubleshooting and support reference

Has IBM MQ for z/OS run successfully before?

Knowing whether IBM MQ for z/OS has successfully run before can help with problem determination, and there are checks you can perform to help you.

If the answer to this question is **No**, consider the following:

- Check your setup.

  If IBM MQ has not run successfully on z/OS before, it is likely that you have not yet set it up correctly. See the information about installing and customizing the queue manager in Installing the IBM MQ for z/OS product for further guidance.
- Verify the installation.
- Check that message CSQ9022I was issued in response to the START QMGR command (indicating normal completion).
- Ensure that z/OS displays IBM MQ as an installed subsystem. To determine if IBM MQ is an installed subsystem use the z/OS command `D OPDATA`.
- Check that the installation verification program (IVP) ran successfully.

- Issue the command `DISPLAY DQM` to check that the channel initiator address space is running, and that the appropriate listeners are started.

Have you applied any APARs or PTFs?

APARs and PTFs can occasionally cause unexpected problems with IBM MQ. These fixes can have been applied to IBM MQ or to other z/OS systems.

If an APAR or PTF has been applied to IBM MQ for z/OS, check that no error message was produced. If the installation was successful, check with the IBM support center for any APAR or PTF error.

If an APAR or PTF has been applied to any other product, consider the effect it might have on the way IBM MQ interfaces with it.

Ensure that you have followed any instructions in the APAR that affect your system. (For example, you might have to redefine a resource.)

Are there any error messages, return codes or other error conditions?

Use this topic to investigate error messages, return codes, and conditions where the queue manager or channel initiator terminated.

The problem might produce the following types of error message or return codes:

CSQ messages and reason codes
IBM MQ for z/OS error messages have the prefix CSQ; if you receive any messages with this prefix (for example, in the console log, or the CICS log), see IBM MQ for z/OS messages, completion, and reason codes for an explanation.

Other messages
For messages with a different prefix, look in the appropriate messages and codes topic for a suggested course of action. See "The message keyword" on page 1651 which lists possible message prefixes and appropriate manuals.

Unusual messages
Be aware of unusual messages associated with the startup of IBM MQ for z/OS, or issued while the system was running before the error occurred. Any unusual messages might indicate some system problem that prevented your application from running successfully.

Application MQI return codes
If your application gets a return code indicating that an MQI call has failed, see Return codes for a description of that return code.

Have you received an unexpected error message or return code?

If your application has received an unexpected error message, consider whether the error message has originated from IBM MQ or from another program.

IBM MQ error messages

IBM MQ for z/OS error messages are prefixed with the letters CSQ.

If you get an unexpected IBM MQ error message (for example, in the console log, or the CICS log), see IBM MQ for z/OS messages, completion, and reason codes for an explanation.

IBM MQ for z/OS messages, completion, and reason codes might give you enough information to resolve the problem quickly, or it might redirect you to another manual for further guidance. If you cannot deal with the message, you might have to contact the IBM support center for help.

Non- IBM MQ error messages

If you get an error message from another IBM program, or from the operating system, look in the messages and codes manual from the appropriate library for an explanation of what it means.

In a queue-sharing environment, look for the following error messages:
- XES (prefixed with the letters IXL)
- Db2 (prefixed with the letters DSN)
- RRS (prefixed with the letters ATR)

See “The message keyword” on page 1651 for a list of the most common message prefixes.

Unexpected return codes
If your application has received an unexpected return code from IBM MQ, see Return codes for information about how your application can handle IBM MQ return codes.

Check for error messages

Issue the DISPLAY THREAD(*) command to check if the queue manager is running. For more information about the command, see DISPLAY THREAD. If the queue manager has stopped running, look for any message that might explain the situation. Messages are displayed on the z/OS console, or on your terminal if you are using the operations and control panels. Use the DISPLAY DQM command to see if the channel initiator is working, and the listeners are active. The z/OS command

```
DISPLAY R,L
```

lists messages with outstanding replies. Check to see whether any of these replies are relevant. In some circumstances, for example, when it has used all its active logs, IBM MQ for z/OS waits for operator intervention.

No error messages issued

If no error messages have been issued, perform the following procedure to determine what is causing the problem:

1. Issue the z/OS commands

   ```
   DISPLAY A,xxxxMSTR
   DISPLAY A,xxxxCHIN
   ```

   (where xxxx is the IBM MQ for z/OS subsystem name). If you receive a message telling you that the queue manager or channel initiator has not been found, this message indicates that the subsystem has terminated. This condition could be caused by an abend or by operator shutdown of the system.
2. If the subsystem is running, you receive message IEE105I. This message includes the *CT=nnnn* field, which contains information about the processor time being used by the subsystem. Note the value of this field, and reissue the command.
   - If the *CT=* value has not changed, this indicates that the subsystem is not using any processor time. This could indicate that the subsystem is in a wait state (or that it has no work to do). If you can issue a command like DISPLAY DQM and you get output back, this indicates there is no work to do rather than a hang condition.
   - If the *CT=* value has changed dramatically, and continues to do so over repeated displays, this could indicate that the subsystem is busy or possibly in a loop.
   - If the reply indicates that the subsystem is now not found, this indicates that it was in the process of terminating when the first command was issued. If a dump is being taken, the subsystem might take a while to terminate. A message is produced at the console before terminating.

     To check that the channel initiator is working, issue the DISPLAY DQM command. If the response does not show the channel initiator working this could be because it is getting insufficient resources (like the processor). In this case, use the z/OS monitoring tools, such as RMF, to determine if there is a resource problem. If it is not, restart the channel initiator.

Has the queue manager or channel initiator terminated abnormally?

Look for any messages saying that the queue manager or channel initiator address space has abnormally terminated. If you get a message for which the system action is to terminate IBM MQ, find out whether a system dump was produced, see IBM MQ dumps.

IBM MQ for z/OS might still be running

Consider also that IBM MQ for z/OS might still be running, but only slowly. If it is running slowly, you probably have a performance problem. To confirm this, see Is your application or IBM MQ for z/OS running slowly. Refer to Dealing with performance problems for advice about what to do next.

Has your application or IBM MQ for z/OS stopped processing work?

There are several reasons why your system would unexpectedly stop processing work including problems with the queue manager, the application, z/OS, and the data sets.

There are several reasons why your system would unexpectedly stop processing work. These include:

Queue manager problems
: The queue manager might be shutting down.

Application problems
: An application programming error might mean that the program branches away from its normal processing, or the application might get in a loop. There might also have been an application abend.

IBM MQ problems
: Your queues might have become disabled for MQPUT or MQGET calls, the dead-letter queue might be full, or IBM MQ for z/OS might be in a wait state, or a loop.

z/OS and other system problems
: z/OS might be in a wait state, or CICS or IMS might be in a wait state or a loop. There might be problems at the system or sysplex level that are affecting the queue manager or the channel initiator. For example, excessive paging. It might also indicate DASD problems, or higher priority tasks with high processor usage.

Db2 and RRS problems
: Check that Db2 and RRS are active.

In all cases, carry out the following checks to determine the cause of the problem:

Is there a problem with the IBM MQ queues?

Use this topic for investigating potential problems with IBM MQ queues.

If you suspect that there is a problem affecting the queues on your subsystem, use the operations and control panels to display the system-command input queue.

If the system responds
: If the system responds, then at least one queue is working. In this case, follow the procedure in "Are some of your queues working?" on page 1303.

If the system does not respond
: The problem might be with the whole subsystem. In this instance, try stopping and restarting the queue manager, responding to any error messages that are produced.

: Check for any messages on the console needing action. Resolve any that might affect IBM MQ, such as a request to mount a tape for an archive log. See if other subsystems or CICS regions are affected.

Use the DISPLAY QMGR COMMANDQ command to identify the name of the system command input queue.

If the problem still occurs after restart
Contact your IBM support center for help (see “Contacting IBM Software Support” on page 1658).

Related concepts:

“Are the correct queues defined?” on page 1304
IBM MQ requires certain predefined queues. Problems can occur if these queues are not defined correctly.

“Does the problem affect only remote or cluster queues?” on page 1304
Use this topic for further investigation if the problem only occurs on remote or cluster queues.

“Does the problem affect only shared queues?” on page 1305
Use this topic to investigate possible queue-sharing group issues which can cause problems for shared queues.

Are some of your queues working?:

Use this topic to investigate when problems occur with a subset of your queues.

If you suspect that the problem occurs with only a subset of queues, select the name of a local queue that you think is having problems and perform the following procedures:

Display queue information
Use the DISPLAY QUEUE and DISPLAY QSTATUS commands to display information about the queue.

Is the queue being processed?

- If CURDEPTH is at MAXDEPTH, it might indicate that the queue is not being processed. Check that all applications that use the queue are running normally (for example, check that transactions in your CICS system are running or that applications started in response to Queue Depth High events are running).
- Issue DISPLAY QSTATUS(xx) IPPROCS to see if the queue is open for input. If not, start the application.
- If CURDEPTH is not at MAXDEPTH, check the following queue attributes to ensure that they are correct:
  - If triggering is being used:
    - Is the trigger monitor running?
    - Is the trigger depth too big?
    - Is the process name correct?
    - Have **all** the trigger conditions been met?

      Issue DISPLAY QSTATUS(xx) IPPROCS to see if an application has the same queue open for input. In some triggering scenarios, a trigger message is not produced if the queue is open for input. Stop the application to cause the triggering processing to be invoked.
  - Can the queue be shared? If not, another application (batch, IMS, or CICS) might already have it open for input.
  - Is the queue enabled appropriately for GET and PUT?

Do you have a long-running unit of work?
If CURDEPTH is not zero, but when you attempt to MQGET a message the queue manager replies that there is no message available, issue either DIS QSTATUS(xx) TYPE(HANDLE) to show you information about applications that have the queue open, or issue DIS CONN(xx) to give you more information about an application that is connected to the queue.

How many tasks are accessing the queues?

Issue DISPLAY QSTATUS(xx) OPPROCS IPPROCS to see how many tasks are putting messages on to, and getting messages from the queue. In a queue-sharing environment, check OPPROCS and IPPROCS on each queue manager. Alternatively, use the CMDSCOPE attribute to check all the queue managers. If there are no application processes getting messages from the queue, determine the reason (for example, because the applications need to be started, a connection has been disrupted, or because the MQOPEN call has failed for some reason).

Is this queue a shared queue? Does the problem affect only shared queues?

Check that there is not a problem with the sysplex elements that support shared queues. For example, check that there is not a problem with the IBM MQ-managed Coupling Facility list structure.

Use D XCF, STRUCTURE, STRNAME=ALL to check that the Coupling Facility structures are accessible.

Use D RRS to check that RRS is active.

Is this queue part of a cluster?

Check to see if the queue is part of a cluster (from the CLUSTER or CLUSNL attribute). If it is, verify that the queue manager that hosts the queue is still active in the cluster.

If you cannot solve the problem

Contact your IBM support center for help (see "Contacting IBM Software Support" on page 1658).

Are the correct queues defined?:

IBM MQ requires certain predefined queues. Problems can occur if these queues are not defined correctly.

Check that the system-command input queue, the system-command reply model queue, and the reply-to queue are correctly defined, and that the MQOPEN calls were successful.

If you are using the system-command reply model queue, check that it was defined correctly.

If you are using clusters, you need to define the SYSTEM.CLUSTER.COMMAND.QUEUE to use commands relating to cluster processing.

Does the problem affect only remote or cluster queues?:

Use this topic for further investigation if the problem only occurs on remote or cluster queues.

If the problem affects only remote or cluster queues, check:

Are the remote queues being accessed?

Check that the programs putting messages to the remote queues have run successfully (see "Dealing with incorrect output" on page 1762).

Is the system link active?

Use APPC or TCP/IP commands as appropriate to check whether the link between the two systems is active.

Use PING or OPING for TCP/IP or D NET ID=xxxxx, E for APPC.

Is triggering working?

If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on and that the queue is get-enabled.

Is the channel or listener running?

If necessary, start the channel or the listener manually, or try stopping and restarting the channel. See Configuring distributed queuing for more information.

Look for error messages on the startup of the channel initiator and listener. See IBM MQ for z/OS messages, completion, and reason codes and Configuring distributed queuing to determine the cause.

What is the channel status?
Check the channel status using the DISPLAY CHSTATUS (channel_name) command.

Are your process and channel definitions correct?
Check your process definitions and your channel definitions.

See Configuring distributed queuing for information about how to use distributed queuing, and for information about how to define channels.

Does the problem affect only shared queues?:

Use this topic to investigate possible queue-sharing group issues which can cause problems for shared queues.

If the problem affects only queue-sharing groups, use the VERIFY QSG function of the CSQ5PQSG utility. This command verifies that the Db2 setup is consistent in terms of the bitmap allocation fields, and object definition for the Db2 queue manager, structure, and shared queue objects, and reports details of any inconsistency that is discovered.

The following is an example of a VERIFY QSG report with errors:

```
CSQU501I  VERIFY QSG function requested
CSQU503I  QSG=SQ02, DB2 DSG=DSN710P5, DB2 ssid=DFP5
CSQU517I  XCF group CSQGSQ02 already defined
CSQU520I  Summary information for XCF group CSQGSQ02
CSQU522I  Member=MQ04, state=QUIESCED, system=MV4A
CSQU523I  User data=D4E5F4C15AD4D8F0F4404040C4C5....
CSQU522I  Member=MQ03, state=QUIESCED, system=MV4A
CSQU523I  User data=D4E5F4C15AD4D8F0F3404040C4C6....
CSQU526I  Connected to DB2 DF4A
CSQU572E  Usage map T01_ARRAY_QMGR and DB2 table CSQ.ADMIN_B_QMGR inconsistent
CSQU573E  QMGR MQ04 in table entry 1 not set in usage map
CSQU574E  QMGR 27 in usage map has no entry in table
CSQU572E  Usage map T01_ARRAY_STRUC and DB2 table CSQ.ADMIN_B_STRUCTURE inconsistent
CSQU575E  Structure APPL2 in table entry 4 not set in usage map
CSQU576E  Structure 55 in usage map has no entry in table
CSQU572E  Usage map T03_LH_ARRAY and DB2 table CSQ.OBJ_B_QUEUE inconsistent
CSQU577E  Queue MYSQ in table entry 13 not set in usage map for structure APPL1
CSQU576E  Queue 129 in usage map for structure APPL1 has no entry in table
CSQU528I  Disconnected from DB2 DF4A
CSQU148I  CSQ5PQSG Utility completed, return code=12
```

Does the problem affect specific parts of the network?

Network problems can cause related problems for MQ for z/OS. Use this topic to review possible sources of networks problems.

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote queue manager is not working, the messages cannot flow to a target queue on the target queue manager. Check that the connection between the two systems is available, and that the channel initiator and listener have been started. Use the MQSC `PING CHANNEL` command to check the connection.

Check that messages are reaching the transmission queue, and check the local queue definition of the transmission queue, and any remote queues. Use the MQSC `BYTSSENT` keyword of the `DISPLAY CHSTATUS` command to check that data is flowing along the channel. Use `DISPLAY QLOCAL (XMITQ) CURDEPTH` to check whether there are messages to be sent on the transmission queue. Check for diagnostic messages at both ends of the channel informing you that messages have been sent to the dead-letter queue.

If you are using IBM MQ clusters, check that the clustering definitions have been set up correctly.

Have you made any network-related changes that might account for the problem?

Have you changed any IBM MQ definitions, or any CICS or IMS definitions? Check the triggering attributes of the transmission queue.

Problems that occur at specific times of the day or affect specific users

Use this topic to review IBM MQ problems that occur at specific times of the day or specific groups of users.

If the problem occurs at specific times of day, it might be that it is dependent on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, and so these periods are the times when load-dependent problems are most likely to occur. (If your network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

If you think that your IBM MQ for z/OS system has a performance problem, see "Dealing with performance problems on z/OS" on page 1755.

If the problem only affects some users, is it because some users do not have the correct security authorization? See User IDs for security checking for information about user IDs checked by IBM MQ for z/OS.

Is the problem intermittent or does the problem occur with all z/OS, CICS, or IMS systems?

Review this topic to consider if problems are caused by application interaction or are related to other z/OS systems.

An intermittent problem could be caused by failing to take into account the fact that processes can run independently of each other. For example, a program might issue an **MQGET** call, without specifying WAIT, before an earlier process has completed. You might also encounter this type of problem if your application tries to get a message from a queue while it is in sync point (that is, before it has been committed).

If the problem only occurs when you access a particular z/OS, IMS, or CICS system, consider what is different about this system. Also consider whether any changes have been made to the system that might affect the way it interacts with IBM MQ.

Has the application run successfully before?

Application errors can often be determined by determining if they have run successfully before or if they have produced error messages and unexpected return codes.

If the problem appears to involve one particular application, consider whether the application has run successfully before.

Before you answer `Yes` to this question, consider:

Have any changes been made to the application since it last ran successfully?
If so, it is likely that the error lies somewhere in the new or modified part of the application. Investigate the changes and see if you can find an obvious reason for the problem.

Have all the functions of the application been fully exercised before?

Did problem occur when part of the application that had never been started before was used for the first time? If so, it is likely that the error lies in that part of the application. Try to find out what the application was doing when it failed, and check the source code in that part of the program for errors.

If a program has been run successfully on many previous occasions, check the current queue status and files that were being processed when the error occurred. It is possible that they contain some unusual data value that causes a rarely used path in the program to be invoked.

Does the application check all return codes?
Has your system has been changed, perhaps in a minor way. Check the return codes your application receives as a result of the change. For example:

- Does your application assume that the queues it accesses can be shared? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?
- Have any security profiles been altered? An **MQOPEN** call might fail because of a security violation; can your application recover from the resulting return code?

Does the application expect particular message formats?
If a message with an unexpected message format has been put onto a queue (for example, a message from a queue manager on a different platform), it might require data conversion or another different form of processing.

Does the application run on other IBM MQ for z/OS systems?
Is something different about the way that this queue manager is set up that is causing the problem? For example, have the queues been defined with the same maximum message length, or default priority?

Does the application use the MQSET call to change queue attributes?
Is the application is designed to set a queue to have no trigger, then process some work, then set the queue to have a trigger? The application might have failed before the queue had been reset to have a trigger.

Does the application handle messages that cause an application to fail?
If an application fails because of a corrupted message, the message retrieved is rolled back. The next application might get the same message and fail in the same way. Ensure that applications use the backout count; when the backout count threshold has been reached, the message in question is put onto the backout queue.

If your application has never run successfully before, examine your application carefully to see if you can find any of the following errors:

Translation and compilation problems
Before you look at the code, examine the output from the translator, the compiler or assembler, and the linkage editor, to see if any errors have been reported. If your application fails to

translate, compile/assemble, or link edit into the load library, it also fails to run if you attempt to invoke it. See Developing applications for information about building your application, and for examples of the job control language (JCL) statements required.

Batch and TSO programs
For batch and TSO programs, check that the correct stub has been included. There is one batch stub and two RRS stubs. If you are using RRS, check that you are not using the MQCMIT and MQBACK calls with the CSQBRSTB stub. Use the CSQBRRSI stub if you want to continue using these calls with RRS.

CICS programs
For CICS programs, check that the program, the IBM MQ CICS stub, and the CICS stub have been linked in the correct order. Also, check that your program or transaction is defined to CICS.

IMS programs
For IMS programs, check that the link includes the program, the IBM MQ stub, and the IMS language interface module. Ensure that the correct entry point has been specified. A program that is loaded dynamically from an IMS program must have the stub and language interface module linked also if it is to use IBM MQ.

Possible code problems
If the documentation shows that each step was accomplished without error, consider the coding of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See “Do you have a program error?” on page 1309 for some examples of common errors that cause problems with IBM MQ applications.

Do applications report errors from IBM MQ ?
For example, a queue might not be enabled for "gets". It receives a return code specifying this condition but does not report it. Consider where your applications report any errors or problems.

Have any changes been made since the last successful run?

Recent changes made since the last successful run are often the source of unexpected errors. This topic contains information about some of the changes which can be investigated as part of your problem determination.

When you are considering changes that might recently have been made, think about IBM MQ, and also about the other programs it interfaces with, the hardware, and any new applications. Consider also the possibility that a new application that you don not yet know about might have been run on the system.

Has your initialization procedure been changed?
Consider whether that might be the cause of the problem. Have you changed any data sets, or changed a library definition? Has z/OS been initialized with different parameters? In addition, check for error messages sent to the console during initialization.

Have you changed any queue definitions or security profiles?
Consider whether some of your queues have been altered so that they are members of a cluster. This change might mean that messages arrive from different sources (for example, other queue managers or applications).

Have you changed any definitions in your sysplex that relate to the support and implementation of shared queues?
Consider the effect that changes to such definitions as your sysplex couple data set, or Coupling Facility resource management policy. These changes might have on the operation of shared queues. Also, consider the effect of changes to the Db2 data sharing environment.

Has any of the software on your z/OS system been upgraded to a later release?
Consider whether there are any necessary post-installation or migration activities that you need to perform.

Has your z/OS subsystem name table been changed?
Changes to levels of corequisite software like z/OS or LE might require additional changes to IBM MQ.

Do your applications deal with return codes that they might get as a result of any changes you have made? Ensure that your applications deal with any new return codes that you introduce.

Do you have a program error?

Use this topic to investigate if a program error is causing an IBM MQ problem.

The examples that follow illustrate the most common causes of problems encountered while running IBM MQ programs. Consider the possibility that the problem with your system could be caused by one of these errors.

- Programs issue MQSET to change queue attributes and fail to reset attributes of a queue. For example, setting a queue to NOTRIGGER.
- Making incorrect assumptions about the attributes of a queue. This assumption could include assuming that queues can be opened with MQOPEN when they are MQOPEN-exclusive, and assuming that queues are not part of a cluster when they are.
- Trying to access queues and data without the correct security authorization.
- Linking a program with no stub, or with the wrong stub (for example, a TSO program with the CICS stub). This can cause either a long-running unit of work, or an X'0C4' or other abend.
- Passing incorrect or invalid parameters in an MQI call; if the wrong number of parameters are passed, no attempt can be made to complete the completion code and reason code fields, and the task is abended. (This is an X'0C4' abend.)

  This problem might occur if you attempt to run an application on an earlier version of MQSeries than it was written for, where some of the MQI values are invalid.
- Failing to define the IBM MQ modules to z/OS correctly (this error causes an X'0C4' abend in CSQYASCP).
- Failing to check return codes from MQI requests.

  This problem might occur if you attempt to run an application on a later version of IBM MQ than it was written for, where new return codes have been introduced that are not checked for.
- Failing to open objects with the correct options needed for later MQI calls, for example using the MQOPEN call to open a queue but not specifying the correct options to enable the queue for subsequent MQGET calls.
- Failing to initialize *MsgId* and *CorrelId* correctly.

  This error is especially true for MQGET.
- Using incorrect addresses.
- Using storage before it has been initialized.
- Passing variables with incorrect lengths specified.
- Passing parameters in the wrong order.
- Failing to define the correct security profiles and classes to RACF.

  This might stop the queue manager or prevent you from carrying out any productive work.
- Relying on default MQI options for a ported application.

  For example, z/OS defaults to MQGET and MQPUT in sync point. The distributed-platform default is out of sync point.
- Relying on default behavior at a normal or abnormal end of a portal application.

  On z/OS, a normal end does an implicit MQCMIT and an abnormal end does an implicit rollback.

Has there been an abend?

Use this topic to investigate common causes of abends and the different types of abend that can cause problems.

If your application has stopped running, it can be caused by an abnormal termination (abend).

You are notified of an abend in one of the following places, depending on what type of application you are using:

Batch Your listing shows the abend.

CICS You see a CICS transaction abend message. If your task is a terminal task, this message is displayed on your screen. If your task is not attached to a terminal, the message is displayed on the CICS CSMT log.

IMS In all cases, you see a message at the IBM MQ for IMS master terminal and in the listing of the dependent region involved. If an IMS transaction that had been entered from a terminal was being processed, an error message is also sent to that terminal.

TSO You might see a TSO message with a return code on your screen. (Whether this message is displayed depends on the way your system is set up, and the type of error.)

Common causes of abends

Abends can be caused by the user ending the task being performed before it terminates normally; for example, if you purge a CICS transaction. Abends can also be caused by an error in an application program.

Address space dumps and transaction dumps

For some abends, an address space dump is produced. For CICS transactions, a transaction dump showing the storage areas of interest to the transaction is provided.

- If an application passes some data, the address of which is no longer valid, a dump is sometimes produced in the address space of the user.

  Note: For a batch dump, the dump is formatted and written to SYSUDUMP. For information about SYSUDUMPs, see “SYSUDUMP information” on page 1752. For CICS, a system dump is written to the SYS1.DUMP data sets, as well as a transaction dump being taken.
- If a problem with IBM MQ for z/OS itself causes an abend, an abend code of X'5C6' or X'6C6' is returned, along with an abend reason code. This reason code uniquely describes the cause of the problem. See “IBM MQ for z/OS abends” on page 1728 for information about the abend codes, and see Return codes for an explanation of the reason code.

Abnormal program termination

If your program has terminated abnormally, see “Dealing with program abends” on page 1729.

If your system has terminated abnormally, and you want to analyze the dump produced, see “IBM MQ for z/OS dumps” on page 1735. This section tells you how to format the dump, and how to interpret the data contained in it.

Have you obtained incorrect output?

Use this topic to review any incorrect output you have received.

If you have obtained what you believe to be some incorrect output, consider the following:

Classifying incorrect output
"Incorrect output⌂ might be regarded as any output that you were not expecting. However, use this term with care in the context of problem determination because it might be a secondary effect of some other type of error. For example, looping could be occurring if you get any repetitive output, even though that output is what you expected.

Error messages
IBM MQ also responds to many errors it detects by sending error messages. You might regard these messages as "incorrect output⌂, but they are only symptoms of another type of problem. If you have received an error message from IBM MQ that you were not expecting, refer to “Are there any error messages, return codes or other error conditions?” on page 1300.

Unexpected messages
If your application has not received a message that it was expecting, has received a message containing unexpected or corrupted information, or has received a message that it was not expecting (for example, one that was destined for a different application), refer to “Dealing with incorrect output” on page 1762.

Can you reproduce the problem?

Reproducing the problem can be used to assist problem determination for IBM MQ for z/OS. Use this topic to further isolate the type of problem reproduction.

If you can reproduce the problem, consider the conditions under which you can reproduce it. For example:

Is it caused by a command?
If so, is the command issued from the z/OS console, from CSQUTIL, from a program written to put commands onto the SYSTEM.COMMAND.INPUT queue, or by using the operations and control panels?

Does the command work if it is entered by another method?
If the command works when it is entered at the console, but not otherwise, check that the command server has not stopped, and that the queue definition of the SYSTEM.COMMAND.INPUT queue has not been changed.

Is the command server running?
Issue the command `DIS CMDSERV` to check.

Is it caused by an application?

If so, does it fail in CICS, IMS, TSO, or batch?

Does it fail on all IBM MQ systems, or only on some?

Is an application causing the problem?
Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application to see if it is in error.

Have you failed to receive a response from an MQSC command?

Use this topic for investigating problems where you fail to receive a response from an MQSC command.

If you have issued an MQSC command from an application (and not from a z/OS console), but you have not received a response, consider the subsequent questions:

Is the command server running?

Check that the command server is running, as follows:

1. Use the DISPLAY CMDSERV command at the z/OS console to display the status of the command server.
2. If the command server is not running, start it using the START CMDSERV command.
3. If the command server is running, issue the DISPLAY QUEUE command. Use the name of the system-command input queue and the CURDEPTH and MAXDEPTH attributes to define the data displayed.

   If these values show that the queue is full, and the command server has been started, the messages are not being read from the queue.
4. Try stopping the command server and then restarting it, responding to any error messages that are produced.
5. Issue the display command again to see if it is working now.

Has a reply been sent to the dead-letter queue?

Use the DISPLAY QMGR DEADQ command to find out the name of the system dead-letter queue (if you do not know what it is).

Use this name in the DISPLAY QUEUE command with the CURDEPTH attribute to see if there are any messages on the queue.

The dead-letter queue message header (dead-letter header structure) contains a reason or feedback code describing the problem. (See Reason (MQLONG) for information about the dead-letter header structure.)

Are the queues enabled for PUTs and GETs?

Use the DISPLAY QUEUE command from the console to check, for example, DISPLAY QUEUE(SYSTEM.COMMAND.INPUT) PUT GET.

Is the `WaitInterval` parameter set to a sufficiently long time?

If your MQGET call has timed out, your application receives completion code of 2 and a reason code of 2033 (MQRC_NO_MSG_AVAILABLE). (See WaitInterval (MQLONG) and MQGET - Get message for information about the *WaitInterval* parameter, and completion and reason codes from MQGET.)

Is a sync point required?

If you are using your own application program to put commands onto the system-command input queue, consider whether you must take a sync point.

You must take a sync point after putting messages to a queue, and before attempting to receive reply messages, or use MQPMO_NO_SYNCPOINT when putting them. Unless you have excluded your request message from sync point, you must take a sync point before attempting to receive reply messages.

Are the `MaxDepth` and `MaxMsgL` parameters of your queues set sufficiently high?

See CSQO016E for information about defining the system-command input queue and the reply-to queue.

Are you using the *CorrelId* and *MsgId* parameters correctly?

You must identify the queue and then display the CURDEPTH. Use the DISPLAY QUEUE command from the console (for example, DISPLAY QUEUE (MY.REPLY.QUEUE) CURDEPTH), to see if there are messages on the reply-to queue that you have not received.

Set the values of `MsgId` and `CorrelId` in your application to ensure that you receive all messages from the queue.

The following questions are applicable if you have issued an MQSC command from either a z/OS console (or its equivalent), or an application, but have not received a response:

Is the queue manager still running, or did your command cause an abend?
Look for error messages indicating an abend, and if one occurred, see "IBM MQ for z/OS dumps" on page 1735.

Were any error messages issued?
Check to see if any error messages were issued that might indicate the nature of the error.

See Issuing commands for information about the different methods you can use to enter MQSC commands.

Is your application or IBM MQ for z/OS running slowly?

Slow applications can be caused by the application itself or underlying software including IBM MQ. Use this topic for initial investigations into slow applications.

If your application is running slowly, this could indicate that it is in a loop, or waiting for a resource that is not available.

Is the problem worse at peak system load times?
This could also be caused by a performance problem. Perhaps it is because your system needs tuning, or because it is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to you to occur at some other time.)

Does the problem occur when the system is lightly loaded?
If you find that degrading performance is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly designed application program is probably to blame. This could manifest itself as a problem that only occurs when specific queues are accessed.

Is IBM MQ for z/OS running slowly?

The following symptoms might indicate that IBM MQ for z/OS is running slowly:

- If your system is slow to respond to commands.
- If repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.

You can find guidance on dealing with waits and loops in "Dealing with applications that are running slowly or have stopped on z/OS" on page 1756, and on dealing with performance problems in "Dealing with performance problems on z/OS" on page 1755.

Reason codes and exceptions

You can use the following messages and reason codes to help you solve problems with your IBM MQ components or applications.

- IBM MQ AMQ messages
- "API completion and reason codes"
- "PCF reason codes" on page 1541
- "Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes" on page 1622
- "WCF custom channel exceptions" on page 1627
- z/OS IBM MQ for z/OS messages, completion, and reason codes
- WMQ JMS Exception Messages

API completion and reason codes

For each call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

For more information about the IBM MQ API, see Developing applications, and the reference information in Developing applications reference.

For a full list and explanation of the API reason codes, see "API reason codes" on page 1315.

API completion codes

The following is a list of the completion codes (MQCC) returned by IBM MQ

0: Successful completion (MQCC_OK)

The call completed fully; all output parameters have been set.

The *Reason* parameter always has the value MQRC_NONE in this case.

1: Warning (partial completion) (MQCC_WARNING)

The call completed partially. Some output parameters might have been set in addition to the *CompCode* and *Reason* output parameters.

The *Reason* parameter gives additional information.

2: Call failed (MQCC_FAILED)

The processing of the call did not complete, and the state of the queue manager is normally unchanged; exceptions are specifically noted. Only the *CompCode* and *Reason* output parameters have been set; all other parameters are unchanged.

The reason might be a fault in the application program, or it might be a result of some situation external to the program, for example the application's authority might have been revoked. The *Reason* parameter gives additional information.

Related reference:

"PCF reason codes" on page 1541
Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

"Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes" on page 1622
IBM MQ can use Secure Sockets Layer (SSL) with the various communication protocols. Use this topic to identify the error codes that can be returned by SSL.

"WCF custom channel exceptions" on page 1627
Diagnostic messages are listed in this topic in numeric order, grouped according to the part of the WCF custom channel from which they originate.

Related information:

IBM MQ AMQ messages

z/OS IBM MQ for z/OS messages, completion, and reason codes

API reason codes

The reason code parameter (*Reason*) is a qualification to the completion code parameter (*CompCode*).

If there is no special reason to report, MQRC_NONE is returned. A successful call returns MQCC_OK and MQRC_NONE.

If the completion code is either MQCC_WARNING or MQCC_FAILED, the queue manager always reports a qualifying reason; details are given under each call description.

Where user exit routines set completion codes and reasons, they should adhere to these rules. In addition, any special reason values defined by user exits should be less than zero, to ensure that they do not conflict with values defined by the queue manager. Exits can set reasons already defined by the queue manager, where these are appropriate.

Reason codes also occur in:

- The *Reason* field of the MQDLH structure
- The *Feedback* field of the MQMD structure

The following is a list of reason codes, in numeric order, providing detailed information to help you understand them, including:

- An explanation of the circumstances that have caused the code to be raised
- The associated completion code
- Suggested programmer actions in response to the code

0 (0000) (RC0): MQRC_NONE:

Explanation

The call completed normally. The completion code (*CompCode*) is MQCC_OK.

Completion Code

MQCC_OK

Programmer response

None.

900 (0384) (RC900): MQRC_APPL_FIRST:

Explanation

This is the lowest value for an application-defined reason code returned by a data-conversion exit. Data-conversion exits can return reason codes in the range MQRC_APPL_FIRST through MQRC_APPL_LAST to indicate particular conditions that the exit has detected.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

As defined by the writer of the data-conversion exit.

999 (03E7) (RC999): MQRC_APPL_LAST:

Explanation

This is the highest value for an application-defined reason code returned by a data-conversion exit. Data-conversion exits can return reason codes in the range MQRC_APPL_FIRST through MQRC_APPL_LAST to indicate particular conditions that the exit has detected.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

As defined by the writer of the data-conversion exit.

2001 (07D1) (RC2001): MQRC_ALIAS_BASE_Q_TYPE_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the `BaseQName` in the alias queue definition resolves to a queue that is not a local queue, a local definition of a remote queue, or a cluster queue or a queue in a distribution list contains an alias queue that is pointing to a topic object

Completion Code

MQCC_FAILED

Programmer response

Correct the queue definitions.

This reason code is also used to identify the corresponding event message Alias Base Queue Type Error.

2002 (07D2) (RC2002): MQRC_ALREADY_CONNECTED:

Explanation

An MQCONN or MQCONNX call was issued, but the application is already connected to the queue manager.
- On z/OS, this reason code occurs for batch and IMS applications only; it does not occur for CICS applications.
- On UNIX, IBM i, Linux and Windows, this reason code occurs if the application attempts to create a nonshared handle when a nonshared handle exists for the thread. A thread can have no more than one nonshared handle.
- On UNIX, IBM i, Linux and Windows, this reason code occurs if an MQCONN call is issued from within an MQ channel exit, API Crossing Exit, or Async Consume Callback function, and a shared hConn is bound to this thread.
- On UNIX, IBM i, Linux and Windows, this reason code occurs if an MQCONNX call that does not specify one of the MQCNO_HANDLE_SHARE_* options is issued from within an MQ channel exit, API Crossing Exit, or Async Consume Callback function, and a shared hConn is bound to this thread
- On Windows, MTS objects do not receive this reason code, as additional connections to the queue manager are permitted.

Completion Code

MQCC_WARNING

Programmer response

None. The `Hconn` parameter returned has the same value as was returned for the previous MQCONN or MQCONNX call.

An MQCONN or MQCONNX call that returns this reason code does *not* mean that an additional MQDISC call must be issued to disconnect from the queue manager. If this reason code is returned because the application has been called in a situation where the MQCONN has already been done, do *not* issue a corresponding MQDISC, because this causes the application that issued the original MQCONN or MQCONNX call to be disconnected as well.

2003 (07D3) (RC2003): MQRC_BACKED_OUT:

Explanation

The current unit of work encountered an unrecoverable error or was backed out. This reason code is issued in the following cases:

- On an MQCMIT or MQDISC call, when the commit operation fails and the unit of work is backed out. All resources that participated in the unit of work are returned to their state at the start of the unit of work. The MQCMIT or MQDISC call completes with MQCC_WARNING in this case.
  - On z/OS, this reason code occurs only for batch applications.
- On an MQGET, MQPUT, or MQPUT1 call that is operating within a unit of work, when the unit of work already encountered an error that prevents the unit of work from being committed (for example, when the log space is exhausted). The application must issue the appropriate call to back out the unit of work. (For a unit of work that is coordinated by the queue manager, this call is the MQBACK call, although the MQCMIT call has the same effect in these circumstances.) The MQGET, MQPUT, or MQPUT1 call completes with MQCC_FAILED in this case.
  - On z/OS, this case does not occur.
- On an asynchronous consumption callback (registered by an MQCB call), the unit of work is backed out and the asynchronous consumer should call MQBACK.
  - On z/OS, this case does not occur.
- For the IBM MQ client on HP Integrity NonStop Server using TMF, this return code can occur:
  - For MQGET, MQPUT, and MQPUT1 calls, if you have an active transaction that is being coordinated by TMF, but the IBM MQ part of the transaction is rolled back because of inactivity on the transaction.
  - If the TMF/Gateway detects that TMF is rolling back the current transaction before the application finishes with it.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Check the returns from previous calls to the queue manager. For example, a previous MQPUT call might have failed.

2004 (07D4) (RC2004): MQRC_BUFFER_ERROR:

Explanation

The *Buffer* parameter is not valid for one of the following reasons:

- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The parameter pointer points to storage that cannot be accessed for the entire length specified by *BufferLength*.
- For calls where *Buffer* is an output parameter: the parameter pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR:

Explanation

The *BufferLength* parameter is not valid, or the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason can also be returned to an MQ MQI client program on the MQCONN or MQCONNX call if the negotiated maximum message size for the channel is smaller than the fixed part of any call structure.

This reason should also be returned by the MQZ_ENUMERATE_AUTHORITY_DATA installable service component when the *AuthorityBuffer* parameter is too small to accommodate the data to be returned to the invoker of the service component.

This reason code can also be returned when a zero length multicast message has been supplied where a positive length is required.

Completion Code

MQCC_FAILED

Programmer response

Specify a value that is zero or greater. For the mqAddString and mqSetString calls, the special value MQBL_NULL_TERMINATED is also valid.

2006 (07D6) (RC2006): MQRC_CHAR_ATTR_LENGTH_ERROR:

Explanation

CharAttrLength is negative (for MQINQ or MQSET calls), or is not large enough to hold all selected attributes (MQSET calls only). This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Specify a value large enough to hold the concatenated strings for all selected attributes.

2007 (07D7) (RC2007): MQRC_CHAR_ATTRS_ERROR:

Explanation

CharAttrs is not valid. The parameter pointer is not valid, or points to read-only storage for MQINQ calls or to storage that is not as long as implied by *CharAttrLength*. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2008 (07D8) (RC2008): MQRC_CHAR_ATTRS_TOO_SHORT:

Explanation

For MQINQ calls, *CharAttrLength* is not large enough to contain all of the character attributes for which MQCA_* selectors are specified in the *Selectors* parameter.

The call still completes, with the *CharAttrs* parameter string filled in with as many character attributes as there is room for. Only complete attribute strings are returned: if there is insufficient space remaining to accommodate an attribute in its entirety, that attribute and subsequent character attributes are omitted. Any space at the end of the string not used to hold an attribute is unchanged.

An attribute that represents a set of values (for example, the namelist *Names* attribute) is treated as a single entity—either all of its values are returned, or none.

Completion Code

MQCC_WARNING

Programmer response

Specify a large enough value, unless only a subset of the values is needed.

2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN:

Explanation

Connection to the queue manager has been lost. This can occur because the queue manager has ended. If the call is an MQGET call with the MQGMO_WAIT option, the wait has been canceled. All connection and object handles are now invalid.

For MQ MQI client applications, it is possible that the call did complete successfully, even though this reason code is returned with a *CompCode* of MQCC_FAILED.

Completion Code

MQCC_FAILED

Programmer response

Applications can attempt to reconnect to the queue manager by issuing the MQCONN or MQCONNX call. It might be necessary to poll until a successful response is received.

- z/OS On z/OS for CICS applications, it is not necessary to issue the MQCONN or MQCONNX call, because CICS applications are connected automatically.

Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

z/OS For z/OS IMS check that the subsystem is started using the IMS DIS SUBSYS command, and if necessary start it using the IMS STA SUBSYS command.

Related information:

IBM MQ and IMS

2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR:

Explanation

The *DataLength* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason can also be returned to an MQ MQI client program on the MQGET, MQPUT, or MQPUT1 call, if the *BufferLength* parameter exceeds the maximum message size that was negotiated for the client channel.

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

If the error occurs for an MQ MQI client program, also check that the maximum message size for the channel is big enough to accommodate the message being sent; if it is not big enough, increase the maximum message size for the channel.

2011 (07DB) (RC2011): MQRC_DYNAMIC_Q_NAME_ERROR:

Explanation

On the MQOPEN call, a model queue is specified in the *ObjectName* field of the *ObjDesc* parameter, but the *DynamicQName* field is not valid, for one of the following reasons:
- *DynamicQName* is completely blank (or blank up to the first null character in the field).
- Characters are present that are not valid for a queue name.
- An asterisk is present beyond the 33rd position (and before any null character).
- An asterisk is present followed by characters that are not null and not blank.

This reason code can also sometimes occur when a server application opens the reply queue specified by the *ReplyToQ* and *ReplyToQMgr* fields in the MQMD of a message that the server has just received. In this case the reason code indicates that the application that sent the original message placed incorrect values into the *ReplyToQ* and *ReplyToQMgr* fields in the MQMD of the original message.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid name.

2012 (07DC) (RC2012): MQRC_ENVIRONMENT_ERROR:

Explanation

The call is not valid for the current environment.
- z/OS On z/OS, when one of the following applies:
  - An MQCONN or MQCONNX call was issued, but the application has been linked with an adapter that is not supported in the environment in which the application is running. For example, this can arise when the application is linked with the MQ RRS adapter, but the application is running in a Db2 Stored Procedure address space. RRS is not supported in this environment. Stored Procedures wishing to use the MQ RRS adapter must run in a Db2 WLM-managed Stored Procedure address space.
  - An MQCMIT or MQBACK call was issued, but the application has been linked with the RRS batch adapter CSQBRSTB. This adapter does not support the MQCMIT and MQBACK calls.
  - An MQCMIT or MQBACK call was issued in the CICS or IMS environment.
  - The RRS subsystem is not operational on the z/OS system that ran the application.
  - An MQCTL call with MQOP_START or an MQCB call registering an Event Listener, was issued but the application is not allowed to create a POSIX thread.
  - An IBM MQ classes for Java application has instantiated an MQQueueManager object using the CLIENT transport. The z/OS environment only supports the use of the BINDINGS transport.
- On IBM i, HP Integrity NonStop Server, UNIX systems, and Windows, when one of the following applies:
  - The application is linked to unsupported libraries.
  - The application is linked to the wrong libraries (threaded or nonthreaded).
  - An MQBEGIN, MQCMIT, or MQBACK call was issued, but an external unit-of-work manager is in use. For example, this reason code occurs on Windows when an MTS object is running as a DTC transaction. This reason code also occurs if the queue manager does not support units of work.
  - The MQBEGIN call was issued in an MQ MQI client environment.

- An MQXCLWLN call was issued, but the call did not originate from a cluster workload exit.
- An MQCONNX call was issued specifying the option MQCNO_HANDLE_SHARE_NONE from within an MQ channel exit, an API Exit, or a Callback function. The reason code occurs only if a shared hConn is bound to the application thread.
- An IBM MQ Object is unable to connect fastpath.
- An IBM MQ classes for Java application has created an MQQueueManager object that uses the CLIENT transport, and then called MQQueueManager.begin(). This method can only be called on MQQueueManager objects that use the BINDINGS transport.

- On Windows, when using the managed .NET client, an attempt was made to use one of the unsupported features:
  - Unmanaged channel exits
  - Secure Sockets Layer (SSL)
  - XA Transactions
  - Communications other than TCP/IP
  - Channel compression

Completion Code

MQCC_FAILED

Programmer response

Do one of the following (as appropriate):

- On z/OS:
  - Link the application with the correct adapter.
  - Modify the application to use the SRRCMIT and SRRBACK calls in place of the MQCMIT and MQBACK calls. Alternatively, link the application with the RRS batch adapter CSQBRRSI. This adapter supports MQCMIT and MQBACK in addition to SRRCMIT and SRRBACK.
  - For a CICS or IMS application, issue the appropriate CICS or IMS call to commit or backout the unit of work.
  - Start the RRS subsystem on the z/OS system that is running the application.
  - If your application uses Language Environment (LE) ensure that it uses the DLL interface and it runs with POSIX(ON).
  - Ensure that your application is allowed to use Unix System Services (USS).
  - Ensure that your Connection Factory definitions for local z/OS applications and WebSphere Application Server applications use Transport Type with bindings mode connections.
- In the other environments:
  - Link the application with the correct libraries (threaded or nonthreaded).
  - Remove from the application the call or feature that is not supported.
  - Change your application to run setuid, if you want to run fastpath.

2013 (07DD) (RC2013): MQRC_EXPIRY_ERROR:

Explanation

On an MQPUT or MQPUT1 call, the value specified for the *Expiry* field in the message descriptor MQMD is not valid.

V 8.0.0.4 This reason code is also generated by JMS applications specifying a delivery delay value greater than either the:
- Message expiry time specified by the application, or
- Expiry time set by the **CUSTOM**(*CAPEXPRY*) attribute of the objects used in the resolution of the target queue or topic.

Completion Code

MQCC_FAILED

Programmer response

Specify a value that is greater than zero, or the special value MQEI_UNLIMITED.

V 8.0.0.4 Ensure that the delivery delay specified by JMS applications is less than the:
- Message expiry time specified by the application, or
- Expiry time set by the **CUSTOM**(*CAPEXPRY*) attribute of the objects used in the resolution of the target queue or topic.

2014 (07DE) (RC2014): MQRC_FEEDBACK_ERROR:

Explanation

On an MQPUT or MQPUT1 call, the value specified for the *Feedback* field in the message descriptor MQMD is not valid. The value is not MQFB_NONE, and is outside both the range defined for system feedback codes and the range defined for application feedback codes.

Completion Code

MQCC_FAILED

Programmer response

Specify MQFB_NONE, or a value in the range MQFB_SYSTEM_FIRST through MQFB_SYSTEM_LAST, or MQFB_APPL_FIRST through MQFB_APPL_LAST.

2016 (07E0) (RC2016): MQRC_GET_INHIBITED:

Explanation

MQGET calls are currently inhibited for the queue, or for the queue to which this queue resolves.

Completion Code

MQCC_FAILED

Programmer response

If the system design allows get requests to be inhibited for short periods, retry the operation later.

This reason code is also used to identify the corresponding event message Get Inhibited.

System programmer action

Use ALTER QLOCAL(...) GET(ENABLED) to allow messages to be got.

2017 (07E1) (RC2017): MQRC_HANDLE_NOT_AVAILABLE:

Explanation

An MQOPEN, MQPUT1 or MQSUB call was issued, but the maximum number of open handles allowed for the current task has already been reached. Be aware that when a distribution list is specified on the MQOPEN or MQPUT1 call, each queue in the distribution list uses one handle.

- On z/OS, "task⌂ means a CICS task, a z/OS task, or an IMS-dependent region.

In addition, the MQSUB call allocates two handles when you do not provide an object handle on input.

Completion Code

MQCC_FAILED

Programmer response

Check whether the application is issuing MQOPEN calls without corresponding MQCLOSE calls. If it is, modify the application to issue the MQCLOSE call for each open object as soon as that object is no longer needed.

Also check whether the application is specifying a distribution list containing a large number of queues that are consuming all of the available handles. If it is, increase the maximum number of handles that the task can use, or reduce the size of the distribution list. The maximum number of open handles that a task can use is given by the *MaxHandles* queue manager attribute.

2018 (07E2) (RC2018): MQRC_HCONN_ERROR:

Explanation

The connection handle `Hconn` is not valid, for one of the following reasons:

- The parameter pointer is not valid, or (for the MQCONN or MQCONNX call) points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The value specified was not returned by a preceding MQCONN or MQCONNX call.
- The value specified has been made invalid by a preceding MQDISC call.
- The handle is a shared handle that has been made invalid by another thread issuing the MQDISC call.
- The handle is a shared handle that is being used on the MQBEGIN call (only nonshared handles are valid on MQBEGIN).
- The handle is a nonshared handle that is being used a thread that did not create the handle.
- The call was issued in the MTS environment in a situation where the handle is not valid (for example, passing the handle between processes or packages; note that passing the handle between library packages *is* supported).
- The conversion program is not defined as OPENAPI, when the MQXCNVC call is invoked by running a character conversion exit program with CICS TS 3.2 or higher. When the conversion process runs, the TCB is switched to the Quasi Reentrant (QR) TCB, making the connection incorrect.

Completion Code

MQCC_FAILED

Programmer response

Ensure that a successful MQCONN or MQCONNX call is performed for the queue manager, and that an MQDISC call has not already been performed for it. Ensure that the handle is being used within its valid scope (see the description of MQCONN in MQCONN for more information about MQCONN).

- On z/OS, also check that the application has been linked with the correct stub; this is CSQCSTUB for CICS applications, CSQBSTUB for batch applications, and CSQQSTUB for IMS applications. Also, the stub used must not belong to a release of the queue manager that is more recent than the release on which the application will run.

Ensure the character conversion exit program run by your CICS TS 3.2 or higher application, which invokes the MQXCNVC call, is defined as OPENAPI. This definition prevents the 2018 MQRC_HCONN_ERROR error caused by from an incorrect connection, and allows the MQGET to complete.

2019 (07E3) (RC2019): MQRC_HOBJ_ERROR:

Explanation

The object handle *Hobj* is not valid, for one of the following reasons:
- The parameter pointer is not valid, or (for the MQOPEN call) points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The value specified was not returned by a preceding MQOPEN call.
- The value specified has been made invalid by a preceding MQCLOSE call.
- The handle is a shared handle that has been made invalid by another thread issuing the MQCLOSE call.
- The handle is a nonshared handle that is being used by a thread that did not create the handle.
- The call is MQGET or MQPUT, but the object represented by the handle is not a queue.

Completion Code

MQCC_FAILED

Programmer response

Ensure that a successful MQOPEN call is performed for this object, and that an MQCLOSE call has not already been performed for it. Ensure that the handle is being used within its valid scope (see the description of MQOPEN in MQOPEN for more information).

2020 (07E4) (RC2020): MQRC_INHIBIT_VALUE_ERROR:

Explanation

On an MQSET call, the value specified for either the MQIA_INHIBIT_GET attribute or the MQIA_INHIBIT_PUT attribute is not valid.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value for the *InhibitGet* or *InhibitPut* queu attribute.

2021 (07E5) (RC2021): MQRC_INT_ATTR_COUNT_ERROR:

Explanation

On an MQINQ or MQSET call, the *IntAttrCount* parameter is negative (MQINQ or MQSET), or smaller than the number of integer attribute selectors (MQIA_*) specified in the *Selectors* parameter (MQSET only). This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Specify a value large enough for all selected integer attributes.

2022 (07E6) (RC2022): MQRC_INT_ATTR_COUNT_TOO_SMALL:

Explanation

On an MQINQ call, the *IntAttrCount* parameter is smaller than the number of integer attribute selectors (MQIA_*) specified in the *Selectors* parameter.

The call completes with MQCC_WARNING, with the *IntAttrs* array filled in with as many integer attributes as there is room for.

Completion Code

MQCC_WARNING

Programmer response

Specify a large enough value, unless only a subset of the values is needed.

2023 (07E7) (RC2023): MQRC_INT_ATTRS_ARRAY_ERROR:

Explanation

On an MQINQ or MQSET call, the *IntAttrs* parameter is not valid. The parameter pointer is not valid (MQINQ and MQSET), or points to read-only storage or to storage that is not as long as indicated by the *IntAttrCount* parameter (MQINQ only). (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED:

Explanation

An MQGET, MQPUT, or MQPUT1 call failed because it would have caused the number of uncommitted messages in the current unit of work to exceed the limit defined for the queue manager (see the `MaxUncommittedMsgs` queue-manager attribute). The number of uncommitted messages is the sum of the following since the start of the current unit of work:

- Messages put by the application with the MQPMO_SYNCPOINT option
- Messages retrieved by the application with the MQGMO_SYNCPOINT option
- Trigger messages and COA report messages generated by the queue manager for messages put with the MQPMO_SYNCPOINT option
- COD report messages generated by the queue manager for messages retrieved with the MQGMO_SYNCPOINT option
- On HP Integrity NonStop Server , this reason code occurs when the maximum number of I/O operations in a single TM/MP transaction has been exceeded.

When publishing messages out of syncpoint to topics it is possible to receive this reason code; see Publications under syncpoint for more information.

Completion Code

MQCC_FAILED

Programmer response

Check whether the application is looping. If it is not, consider reducing the complexity of the application. Alternatively, increase the queue-manager limit for the maximum number of uncommitted messages within a unit of work.

- On z/OS, the limit for the maximum number of uncommitted messages can be changed by using the ALTER QMGR command.
- On IBM i , the limit for the maximum number of uncommitted messages can be changed by using the CHGMQM command.
- On HP Integrity NonStop Server , the application should cancel the transaction and retry with a smaller number of operations in the unit of work. See the *MQSeries for Tandem NonStop Kernel System Management Guide* for more details.

2025 (07E9) (RC2025): MQRC_MAX_CONNS_LIMIT_REACHED:

Explanation

The MQCONN or MQCONNX call was rejected because the maximum number of concurrent connections has been exceeded.

- On z/OS, the connection limits are 32767 for both TSO and Batch.
- On IBM i, HP Integrity NonStop Server, UNIX systems, and Windows, this reason code can also occur on the MQOPEN call.
- When using Java applications, the connection manager might define a limit to the number of concurrent connections.

  Note: The application using IBM MQ might have delegated the management of connections to a framework or connection pool, for example, a Java EE application server, an application framework such as Spring, an IBM Container (for IBM Cloud (formerly Bluemix®)), or a combination of these. For more information, see Use JMS connection pools.

Completion Code

MQCC_FAILED

Programmer response

Either increase the size of the appropriate parameter value, or reduce the number of concurrent connections.

Related information:

Connection pooling in IBM MQ classes for Java

2026 (07EA) (RC2026): MQRC_MD_ERROR:

Explanation

The MQMD structure is not valid, for one of the following reasons:

- The *StrucId* field is not MQMD_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer response

Ensure that input fields in the MQMD structure are set correctly.

2027 (07EB) (RC2027): MQRC_MISSING_REPLY_TO_Q:

Explanation

On an MQPUT or MQPUT1 call, the *ReplyToQ* field in the message descriptor MQMD is blank, but one or both of the following is true:

- A reply was requested (that is, MQMT_REQUEST was specified in the *MsgType* field of the message descriptor).
- A report message was requested in the *Report* field of the message descriptor.

Completion Code

MQCC_FAILED

Programmer response

Specify the name of the queue to which the reply message or report message is to be sent.

2029 (07ED) (RC2029): MQRC_MSG_TYPE_ERROR:

Explanation

Either:

- On an MQPUT or MQPUT1 call, the value specified for the *MsgType* field in the message descriptor (MQMD) is not valid.
- A message processing program received a message that does not have the expected message type. For example, if the IBM MQ command server receives a message which is not a request message (MQMT_REQUEST) then it rejects the request with this reason code.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value for the *MsgType* field. In the case where a request is rejected by a message processing program, refer to the documentation for that program for details of the message types that it supports.

2030 (07EE) (RC2030): MQRC_MSG_TOO_BIG_FOR_Q:

Explanation

An MQPUT or MQPUT1 call was issued to put a message on a queue, but the message was too long for the queue and MQMF_SEGMENTATION_ALLOWED was not specified in the *MsgFlags* field in MQMD. If segmentation is not allowed, the length of the message cannot exceed the lesser of the queue *MaxMsgLength* attribute and queue-manager *MaxMsgLength* attribute.

- On z/OS, the queue manager does not support the segmentation of messages; if MQMF_SEGMENTATION_ALLOWED is specified, it is accepted but ignored.

This reason code can also occur when MQMF_SEGMENTATION_ALLOWED *is* specified, but the nature of the data present in the message prevents the queue manager splitting it into segments that are small enough to place on the queue:

- For a user-defined format, the smallest segment that the queue manager can create is 16 bytes.
- For a built-in format, the smallest segment that the queue manager can create depends on the particular format, but is greater than 16 bytes in all cases other than MQFMT_STRING (for MQFMT_STRING the minimum segment size is 16 bytes).

MQRC_MSG_TOO_BIG_FOR_Q can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code

MQCC_FAILED

Programmer response

Check whether the *BufferLength* parameter is specified correctly; if it is, do one of the following:

- Increase the value of the queue's *MaxMsgLength* attribute; the queue-manager's *MaxMsgLength* attribute may also need increasing.
- Break the message into several smaller messages.

- Specify MQMF_SEGMENTATION_ALLOWED in the *MsgFlags* field in MQMD; this will allow the queue manager to break the message into segments.

2031 (07EF) (RC2031): MQRC_MSG_TOO_BIG_FOR_Q_MGR:

Explanation

An MQPUT or MQPUT1 call was issued to put a message on a queue, but the message was too long for the queue manager and MQMF_SEGMENTATION_ALLOWED was not specified in the *MsgFlags* field in MQMD. If segmentation is not allowed, the length of the message cannot exceed the lesser of the queue-manager *MaxMsgLength* attribute and queue *MaxMsgLength* attribute.

This reason code can also occur when MQMF_SEGMENTATION_ALLOWED *is* specified, but the nature of the data present in the message prevents the queue manager splitting it into segments that are small enough for the queue-manager limit:
- For a user-defined format, the smallest segment that the queue manager can create is 16 bytes.
- For a built-in format, the smallest segment that the queue manager can create depends on the particular format, but is greater than 16 bytes in all cases other than MQFMT_STRING (for MQFMT_STRING the minimum segment size is 16 bytes).

MQRC_MSG_TOO_BIG_FOR_Q_MGR can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

This reason also occurs if a channel, through which the message is to pass, has restricted the maximum message length to a value that is actually less than that supported by the queue manager, and the message length is greater than this value.
- On z/OS, this return code is issued only if you are using CICS for distributed queuing. Otherwise, MQRC_MSG_TOO_BIG_FOR_CHANNEL is issued.

Completion Code

MQCC_FAILED

Programmer response

Check whether the *BufferLength* parameter is specified correctly; if it is, do one of the following:
- Increase the value of the queue-manager's *MaxMsgLength* attribute; the queue's *MaxMsgLength* attribute may also need increasing.
- Break the message into several smaller messages.
- Specify MQMF_SEGMENTATION_ALLOWED in the *MsgFlags* field in MQMD; this will allow the queue manager to break the message into segments.
- Check the channel definitions.

2033 (07F1) (RC2033): MQRC_NO_MSG_AVAILABLE:

Explanation

An MQGET call was issued, but there is no message on the queue satisfying the selection criteria specified in MQMD (the *MsgId* and *CorrelId* fields), and in MQGMO (the *Options* and *MatchOptions* fields). Either the MQGMO_WAIT option was not specified, or the time interval specified by the *WaitInterval* field in MQGMO has expired. This reason is also returned for an MQGET call for browse, when the end of the queue has been reached.

This reason code can also be returned by the mqGetBag and mqExecute calls. mqGetBag is similar to MQGET. For the mqExecute call, the completion code can be either MQCC_WARNING or MQCC_FAILED:

- If the completion code is MQCC_WARNING, some response messages were received during the specified wait interval, but not all. The response bag contains system-generated nested bags for the messages that were received.
- If the completion code is MQCC_FAILED, no response messages were received during the specified wait interval.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

If this is an expected condition, no corrective action is required.

If this is an unexpected condition, check that:

- The message was put on the queue successfully.
- The unit of work (if any) used for the MQPUT or MQPUT1 call was committed successfully.
- The options controlling the selection criteria are specified correctly. All of the following can affect the eligibility of a message for return on the MQGET call:
  - MQGMO_LOGICAL_ORDER
  - MQGMO_ALL_MSGS_AVAILABLE
  - MQGMO_ALL_SEGMENTS_AVAILABLE
  - MQGMO_COMPLETE_MSG
  - MQMO_MATCH_MSG_ID
  - MQMO_MATCH_CORREL_ID
  - MQMO_MATCH_GROUP_ID
  - MQMO_MATCH_MSG_SEQ_NUMBER
  - MQMO_MATCH_OFFSET
  - Value of *MsgId* field in MQMD
  - Value of *CorrelId* field in MQMD

Consider waiting longer for the message.

2034 (07F2) (RC2034): MQRC_NO_MSG_UNDER_CURSOR:

Explanation

An MQGET call was issued with either the MQGMO_MSG_UNDER_CURSOR or the MQGMO_BROWSE_MSG_UNDER_CURSOR option. However, the browse cursor is not positioned at a retrievable message. This is caused by one of the following:

- The cursor is positioned logically before the first message (as it is before the first MQGET call with a browse option has been successfully performed).
- The message the browse cursor was positioned on has been locked or removed from the queue (probably by some other application) since the browse operation was performed.
- The message the browse cursor was positioned on has expired.

Completion Code

MQCC_FAILED

Programmer response

Check the application logic. This may be an expected reason if the application design allows multiple servers to compete for messages after browsing. Consider also using the MQGMO_LOCK option with the preceding browse MQGET call.

2035 (07F3) (RC2035): MQRC_NOT_AUTHORIZED:
General explanation

Explanation

The user of the application or channel that produced the error, is not authorized to perform the operation attempted:

- On an MQCONN or MQCONNX call, the user is not authorized to connect to the queue manager.
  - On z/OS, for CICS applications, MQRC_CONNECTION_NOT_AUTHORIZED is issued instead.
- On an MQCONNX call, the length of the user ID or password is greater than the maximum length permitted. The maximum length of the user ID is dependent on the platform. For more information, see User IDs.
- On an MQOPEN or MQPUT1 call, the user is not authorized to open the object for the option(s) specified.
  - On z/OS, if the object being opened is a model queue, this reason also arises if the user is not authorized to create a dynamic queue with the required name.
- On an MQCLOSE call, the user is not authorized to delete the object, which is a permanent dynamic queue, and the `Hobj` parameter specified on the MQCLOSE call is not the handle returned by the MQOPEN call that created the queue.
- On a command, the user is not authorized to issue the command, or to access the object it specifies.
- On an MQSUB call, the user is not authorized to subscribe to the topic.
- On an MQSUB call, using non-managed destination queues, the user is not authorized to use the destination queue.

This reason code can also occur in the `Feedback` field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct queue manager or object was specified, and that appropriate authority exists.

This reason code is also used to identify the corresponding event message,
- MQCONN or MQCONNX Not Authorized (type 1).
- MQOPEN or MQPUT1 Not Authorized (type 2).
- MQCLOSE Not Authorized (type 3).
- Command Not Authorized (type 4).
- MQSUB Not Authorized (type 5).
- MQSUB destination Not Authorized (type 6).

Specific problems generating RC2035

JMSWMQ2013 invalid security authentication

See Invalid security authentication for information when your IBM MQ JMS application fails with security authentication errors.

MQRC_NOT_AUTHORIZED on a queue or channel

See MQRC_NOT_AUTHORIZED on a queue for information when MQRC 2035 (MQRC_NOT_AUTHORIZED) is returned where a user is not authorized to perform the function. Determine which object the user cannot access and provide the user access to the object.

MQRC_NOT_AUTHORIZED (AMQ4036 on a client) as an administrator

See MQRC_NOT_AUTHORIZED as an administrator for information when MQRC 2035 (MQRC_NOT_AUTHORIZED) is returned where you try to use a user ID that is an IBM MQ Administrator, to remotely access the queue manager through a client connection.

MQS_REPORT_NOAUTH

See MQS_REPORT_NOAUTH for information on using this environment variable to better diagnose return code 2035 (MQRC_NOT_AUTHORIZED). The use of this environment variable generates errors in the queue manager error log, but does not generate a Failure Data Capture (FDC).

MQSAUTHERRORS

See MQSAUTHERRORS for information on using this environment variable to generate FDC files related to return code 2035 (MQRC_NOT_AUTHORIZED). The use of this environment variable generates an FDC, but does not generate errors in the queue manager error log.

2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE:

Explanation

An MQGET call was issued with one of the following options:
- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MSG_UNDER_CURSOR

but either the queue had not been opened for browse, or you are using IBM MQ Multicast messaging.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_BROWSE when the queue is opened.

If you are using IBM MQ Multicast messaging, you cannot specify browse options with an MQGET call.

2037 (07F5) (RC2037): MQRC_NOT_OPEN_FOR_INPUT:

Explanation

An MQGET call was issued to retrieve a message from a queue, but the queue had not been opened for input.

Completion Code

MQCC_FAILED

Programmer response

Specify one of the following when the queue is opened:
- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_AS_Q_DEF

2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE:

Explanation

An MQINQ call was issued to inquire object attributes, but the object had not been opened for inquire.

An MQINQ call was issued for a topic handle in IBM MQ Multicast.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_INQUIRE when the object is opened.

MQINQ is not supported for topic handles in IBM MQ Multicast.

2039 (07F7) (RC2039): MQRC_NOT_OPEN_FOR_OUTPUT:

Explanation

An MQPUT call was issued to put a message on a queue, but the queue had not been opened for output.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_OUTPUT when the queue is opened.

2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET:

Explanation

An MQSET call was issued to set queue attributes, but the queue had not been opened for set.

An MQSET call was issued for a topic handle in IBM MQ Multicast.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_SET when the object is opened.

MQSET is not supported for topic handles in IBM MQ Multicast.

2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED:

Explanation

Object definitions that affect this object have been changed since the `Hobj` handle used on this call was returned by the MQOPEN call. For more information about the MQOPEN call, see MQOPEN.

This reason does not occur if the object handle is specified in the `Context` field of the `PutMsgOpts` parameter on the MQPUT or MQPUT1 call.

Completion Code

MQCC_FAILED

Programmer response

Issue an MQCLOSE call to return the handle to the system. It is then usually sufficient to reopen the object and retry the operation. However, if the object definitions are critical to the application logic, an MQINQ call can be used after reopening the object, to obtain the new values of the object attributes.

2042 (07FA) (RC2042): MQRC_OBJECT_IN_USE:

Explanation

An MQOPEN call was issued, but the object in question has already been opened by this or another application with options that conflict with those specified in the `Options` parameter. This arises if the request is for shared input, but the object is already open for exclusive input; it also arises if the request is for exclusive input, but the object is already open for input (of any sort).

MCAs for receiver channels, or the intra-group queuing agent (IGQ agent), may keep the destination queues open even when messages are not being transmitted; this results in the queues appearing to be "in use". Use the MQSC command DISPLAY QSTATUS to find out who is keeping the queue open.
- On z/OS, this reason can also occur for an MQOPEN or MQPUT1 call, if the object to be opened (which can be a queue, or for MQOPEN a namelist or process object) is in the process of being deleted.

Completion Code

MQCC_FAILED

Programmer response

System design should specify whether an application is to wait and retry, or take other action.

2043 (07FB) (RC2043): MQRC_OBJECT_TYPE_ERROR:

Explanation

On the MQOPEN or MQPUT1 call, the *ObjectType* field in the object descriptor MQOD specifies a value that is not valid. For the MQPUT1 call, the object type must be MQOT_Q.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid object type.

2044 (07FC) (RC2044): MQRC_OD_ERROR:

Explanation

On the MQOPEN or MQPUT1 call, the object descriptor MQOD is not valid, for one of the following reasons:
- The *StrucId* field is not MQOD_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer response

Ensure that input fields in the MQOD structure are set correctly.

2045 (07FD) (RC2045): MQRC_OPTION_NOT_VALID_FOR_TYPE:

Explanation

On an MQOPEN or MQCLOSE call, an option is specified that is not valid for the type of object or queue being opened or closed.

For the MQOPEN call, this includes the following cases:
- An option that is inappropriate for the object type (for example, MQOO_OUTPUT for an MQOT_PROCESS object).
- An option that is unsupported for the queue type (for example, MQOO_INQUIRE for a remote queue that has no local definition).
- One or more of the following options:
  - MQOO_INPUT_AS_Q_DEF
  - MQOO_INPUT_SHARED
  - MQOO_INPUT_EXCLUSIVE
  - MQOO_BROWSE

  - MQOO_INQUIRE
  - MQOO_SET

  when either:

  - the queue name is resolved through a cell directory, or
  - *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (to specify a queue-manager alias), and the queue named in the *RemoteQMgrName* attribute of the definition is the name of the local queue manager.

For the MQCLOSE call, this includes the following case:

- The MQCO_DELETE or MQCO_DELETE_PURGE option when the queue is not a dynamic queue.

This reason code can also occur on the MQOPEN call when the object being opened is of type MQOT_NAMELIST, MQOT_PROCESS, or MQOT_Q_MGR, but the *ObjectQMgrName* field in MQOD is neither blank nor the name of the local queue manager.

Completion Code

MQCC_FAILED

Programmer response

Specify the correct option. For the MQOPEN call, ensure that the *ObjectQMgrName* field is set correctly. For the MQCLOSE call, either correct the option or change the definition type of the model queue that is used to create the new queue.

2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR:

Explanation

The *Options* parameter or field contains options that are not valid, or a combination of options that is not valid.

- For the MQOPEN, MQCLOSE, MQXCNVC, mqBagToBuffer, mqBufferToBag, mqCreateBag, and mqExecute calls, *Options* is a separate parameter on the call.

  This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- For the MQBEGIN, MQCONNX, MQGET, MQPUT, and MQPUT1 calls, *Options* is a field in the relevant options structure (MQBO, MQCNO, MQGMO, or MQPMO).
- For more information about option errors for IBM MQ Multicast see: MQI concepts and how they relate to multicast.

Completion Code

MQCC_FAILED

Programmer response

Specify valid options. Check the description of the *Options* parameter or field to determine which options and combinations of options are valid. If multiple options are being set by adding the individual options together, ensure that the same option is not added twice. For more information, see Rules for validating MQI options.

2047 (07FF) (RC2047): MQRC_PERSISTENCE_ERROR:

Explanation

On an MQPUT or MQPUT1 call, the value specified for the *Persistence* field in the message descriptor MQMD is not valid.

Completion Code

MQCC_FAILED

Programmer response

Specify one of the following values:
- MQPER_PERSISTENT
- MQPER_NOT_PERSISTENT
- MQPER_PERSISTENCE_AS_Q_DEF

2048 (0800) (RC2048): MQRC_PERSISTENT_NOT_ALLOWED:

Explanation

On an MQPUT or MQPUT1 call, the value specified for the *Persistence* field in MQMD (or obtained from the *DefPersistence* queue attribute) specifies MQPER_PERSISTENT, but the queue on which the message is being placed does not support persistent messages. Persistent messages cannot be placed on temporary dynamic queues.

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code

MQCC_FAILED

Programmer response

Specify MQPER_NOT_PERSISTENT if the message is to be placed on a temporary dynamic queue. If persistence is required, use a permanent dynamic queue or predefined queue in place of a temporary dynamic queue.

Be aware that server applications are recommended to send reply messages (message type MQMT_REPLY) with the same persistence as the original request message (message type MQMT_REQUEST). If the request message is persistent, the reply queue specified in the *ReplyToQ* field in the message descriptor MQMD cannot be a temporary dynamic queue. Use a permanent dynamic queue or predefined queue as the reply queue in this situation.

On z/OS, you cannot put persistent messages to a shared queue if the CFSTRUCT that the queue uses is defined with RECOVER(NO). Either put only non-persistent messages to this queue or change the CFSTRUCT definition to RECOVER(YES). If you put a persistent message to a queue that uses a CFSTRUCT with RECOVER(NO) the put will fail with MQRC_PERSISTENT_NOT_ALLOWED.

2049 (0801) (RC2049): MQRC_PRIORITY_EXCEEDS_MAXIMUM:

Explanation

An MQPUT or MQPUT1 call was issued, but the value of the *Priority* field in the message descriptor MQMD exceeds the maximum priority supported by the local queue manager, as shown by the *MaxPriority* queue-manager attribute. The message is accepted by the queue manager, but is placed on the queue at the queue manager's maximum priority. The *Priority* field in the message descriptor retains the value specified by the application that put the message.

Completion Code

MQCC_WARNING

Programmer response

None required, unless this reason code was not expected by the application that put the message.

2050 (0802) (RC2050): MQRC_PRIORITY_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the value of the *Priority* field in the message descriptor MQMD is not valid. The maximum priority supported by the queue manager is given by the *MaxPriority* queue-manager attribute.

Completion Code

MQCC_FAILED

Programmer response

Specify a value in the range zero through *MaxPriority*, or the special value MQPRI_PRIORITY_AS_Q_DEF.

2051 (0803) (RC2051): MQRC_PUT_INHIBITED:

Explanation

MQPUT and MQPUT1 calls are currently inhibited for the queue, or for the queue to which this queue resolves.

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code

MQCC_FAILED

Programmer response

If the system design allows put requests to be inhibited for short periods, retry the operation later.

This reason code is also used to identify the corresponding event message Put Inhibited.

System programmer action

Use ALTER QLOCAL(...) PUT(ENABLED) to allow messages to be put.

2052 (0804) (RC2052): MQRC_Q_DELETED:

Explanation

An *Hobj* queue handle specified on a call refers to a dynamic queue that has been deleted since the queue was opened. For more information about the deletion of dynamic queues, see the description of MQCLOSE in MQCLOSE.

- On z/OS, this can also occur with the MQOPEN and MQPUT1 calls if a dynamic queue is being opened, but the queue is in a logically-deleted state. See MQCLOSE for more information about this.

Completion Code

MQCC_FAILED

Programmer response

Issue an MQCLOSE call to return the handle and associated resources to the system (the MQCLOSE call will succeed in this case). Check the design of the application that caused the error.

2053 (0805) (RC2053): MQRC_Q_FULL:

Explanation

An MQPUT or MQPUT1 call, or a command, failed because the queue is full, that is, it already contains the maximum number of messages possible, as specified by the *MaxQDepth* queue attribute.

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code

MQCC_FAILED

Programmer response

Retry the operation later. Consider increasing the maximum depth for this queue, or arranging for more instances of the application to service the queue.

This reason code is also used to identify the corresponding event message Queue Full.

2055 (0807) (RC2055): MQRC_Q_NOT_EMPTY:

Explanation

An MQCLOSE call was issued for a permanent dynamic queue, but the call failed because the queue is not empty or still in use. One of the following applies:

- The MQCO_DELETE option was specified, but there are messages on the queue.
- The MQCO_DELETE or MQCO_DELETE_PURGE option was specified, but there are uncommitted get or put calls outstanding against the queue.

See the usage notes pertaining to dynamic queues for the MQCLOSE call for more information.

This reason code is also returned from a command to clear or delete or move a queue, if the queue contains uncommitted messages (or committed messages in the case of delete queue without the purge option).

Completion Code

MQCC_FAILED

Programmer response

Check why there might be messages on the queue. Be aware that the *CurrentQDepth* queue attribute might be zero even though there are one or more messages on the queue; this can happen if the messages have been retrieved as part of a unit of work that has not yet been committed. If the messages can be discarded, try using the MQCLOSE call with the MQCO_DELETE_PURGE option. Consider retrying the call later.

2056 (0808) (RC2056): MQRC_Q_SPACE_NOT_AVAILABLE:

Explanation

An MQPUT or MQPUT1 call was issued, but there is no space available for the queue on disk or other storage device.

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Check whether an application is putting messages in an infinite loop. If not, make more disk space available for the queue.

2057 (0809) (RC2057): MQRC_Q_TYPE_ERROR:

Explanation

One of the following occurred:
- On an MQOPEN call, the `ObjectQMgrName` field in the object descriptor MQOD or object record MQOR specifies the name of a local definition of a remote queue (to specify a queue-manager alias), and in that local definition the `RemoteQMgrName` attribute is the name of the local queue manager. However, the `ObjectName` field in MQOD or MQOR specifies the name of a model queue on the local queue manager; this is not allowed. For more information, see MQOPEN.
- On an MQPUT1 call, the object descriptor MQOD or object record MQOR specifies the name of a model queue.
- On a previous MQPUT or MQPUT1 call, the `ReplyToQ` field in the message descriptor specified the name of a model queue, but a model queue cannot be specified as the destination for reply or report messages. Only the name of a predefined queue, or the name of the *dynamic* queue created from the model queue, can be specified as the destination. In this situation the reason code MQRC_Q_TYPE_ERROR is returned in the `Reason` field of the MQDLH structure when the reply message or report message is placed on the dead-letter queue.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid queue.

This reason code is also used to identify the corresponding event message Queue Type Error.

2058 (080A) (RC2058): MQRC_Q_MGR_NAME_ERROR:

Explanation

On an MQCONN or MQCONNX call, the value specified for the `QMgrName` parameter is not valid or not known. This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- z/OS On z/OS for CICS applications, this reason can occur on *any* call if the original connect specified an incorrect or unrecognized name.

z/OS For CICS, this reason can be caused by the wrong resync value. For example, Groupresync is specified and the queue manager is not in a queue sharing group.

This reason code can also occur if an MQ MQI client application attempts to connect to a queue manager within an MQ-client queue-manager group (see the `QMgrName` parameter of MQCONN), and either:
- Queue-manager groups are not supported.
- There is no queue-manager group with the specified name.

z/OS For IMS adapter on z/OS, this can be caused by the library containing your CSQQDEFV module being after the SCSQAUTH data set in steplib. Move your library above the SCSQAUTH data set.

Completion Code

MQCC_FAILED

Programmer response

Use an all-blank name if possible, or verify that the name used is valid.

If you are using CICS Resyncmember(Groupresync), use the queue-sharing group (QSG) name in the MQNAME rather than the queue manager name.

2059 (080B) (RC2059): MQRC_Q_MGR_NOT_AVAILABLE:

Explanation

This error occurs:

1. On an MQCONN or MQCONNX call, the queue manager identified by the *QMgrName* parameter is not available for connection.
   - On z/OS:
     - For batch applications, this reason can be returned to applications running in LPARs that do not have a queue manager installed.
     - For CICS applications, this reason can occur on any call if the original connect specified a queue manager with a name that was recognized, but which is not available.
   - On IBM i, this reason can also be returned by the MQOPEN and MQPUT1 calls, when MQHC_DEF_HCONN is specified for the *Hconn* parameter by an application running in compatibility mode.
2. On an MQCONN or MQCONNX call from an IBM MQ MQI client application:
   - Attempting to connect to a queue manager within an MQ-client queue-manager group when none of the queue managers in the group is available for connection (see the *QMgrName* parameter of the MQCONN call).
   - If the client channel fails to connect, perhaps because of an error with the client-connection or the corresponding server-connection channel definitions.
3. If a command uses the *CommandScope* parameter specifying a queue manager that is not active in the queue-sharing group.
4. In a multiple installation environment, where an application attempts to connect to a queue manager associated with an installation of IBM WebSphere MQ Version 7.1, or later, but has loaded libraries from IBM WebSphere MQ Version 7.0.1. IBM WebSphere MQ Version 7.0.1 cannot load libraries from other versions of IBM MQ.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the queue manager has been started. If the connection is from a client application, check the channel definitions, channel status, and error logs.

In a multiple installation environment, ensure that IBM WebSphere MQ Version 7.1, or later, libraries are loaded by the operating system. For more information, see Connecting applications in a multiple installation environment.

2061 (080D) (RC2061): MQRC_REPORT_OPTIONS_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the *Report* field in the message descriptor MQMD contains one or more options that are not recognized by the local queue manager. The options that cause this reason code to be returned depend on the destination of the message; see the description of REPORT in Report options and message flags for more details.

This reason code can also occur in the *Feedback* field in the MQMD of a report message, or in the *Reason* field in the MQDLH structure of a message on the dead-letter queue; in both cases it indicates that the destination queue manager does not support one or more of the report options specified by the sender of the message.

Completion Code

MQCC_FAILED

Programmer response

Do the following:
- Ensure that the *Report* field in the message descriptor is initialized with a value when the message descriptor is declared, or is assigned a value prior to the MQPUT or MQPUT1 call. Specify MQRO_NONE if no report options are required.
- Ensure that the report options specified are valid; see the *Report* field described in the description of MQMD in Report options and message flags for valid report options.
- If multiple report options are being set by adding the individual report options together, ensure that the same report option is not added twice.
- Check that conflicting report options are not specified. For example, do not add both MQRO_EXCEPTION and MQRO_EXCEPTION_WITH_DATA to the *Report* field; only one of these can be specified.

2062 (080E) (RC2062): MQRC_SECOND_MARK_NOT_ALLOWED:

Explanation

An MQGET call was issued specifying the MQGMO_MARK_SKIP_BACKOUT option in the *Options* field of MQGMO, but a message has already been marked within the current unit of work. Only one marked message is allowed within each unit of work.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Modify the application so that no more than one message is marked within each unit of work.

2063 (080F) (RC2063): MQRC_SECURITY_ERROR:

Explanation

An MQCONN, MQCONNX, MQOPEN, MQSUB, MQPUT1, or MQCLOSE call was issued, but it failed because a security error occurred.

- On z/OS, there are two possible reasons for this:
  - An MQCONN or MQCONNX call was issued to connect to the queue manager using the BINDINGS transport, passing in a username or password, or both, that were longer than 8 characters.
  - The security error was returned by the External Security Manager.
- If you are using AMS, this could be a set up issue.
- If you are using connection authentication with an LDAP server, this could be as a result of connectivity failure to the LDAP server, or an error from the LDAP server.

Completion Code

MQCC_FAILED

Programmer response

Note the error from the security manager, and contact your system programmer or security administrator.

- If you are using AMS, you should check the queue manager error logs.
- On z/OS, ensure that both the username and password specified, when connecting to the queue manager, have a maximum length of 8 characters.
- On IBM i, the FFST log will contain the error information.
- If you are using LDAP, use DISPLAY QMSTATUS to check the status of the connection to the LDAP server, and check the queue manager error logs for any error messages.

2065 (0811) (RC2065): MQRC_SELECTOR_COUNT_ERROR:

Explanation

On an MQINQ or MQSET call, the `SelectorCount` parameter specifies a value that is not valid. This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Specify a value in the range 0 through 256.

2066 (0812) (RC2066): MQRC_SELECTOR_LIMIT_EXCEEDED:

Explanation

On an MQINQ or MQSET call, the *SelectorCount* parameter specifies a value that is larger than the maximum supported (256).

Completion Code

MQCC_FAILED

Programmer response

Reduce the number of selectors specified on the call; the valid range is 0 through 256.

2067 (0813) (RC2067): MQRC_SELECTOR_ERROR:

Explanation

An MQINQ or MQSET call was issued, but the *Selectors* array contains a selector that is not valid for one of the following reasons:

- The selector is not supported or out of range.
- The selector is not applicable to the type of object with attributes that are being inquired upon or set.
- The selector is for an attribute that cannot be set.

This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

An MQINQ call was issued for a managed handle in IBM MQ Multicast, inquiring a value other than *Current Depth*.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the value specified for the selector is valid for the object type represented by *Hobj*. For the MQSET call, also ensure that the selector represents an integer attribute that can be set.

MQINQ for managed handles in IBM MQ Multicast can only inquire on *Current Depth*.

2068 (0814) (RC2068): MQRC_SELECTOR_NOT_FOR_TYPE:

Explanation

On the MQINQ call, one or more selectors in the *Selectors* array is not applicable to the type of the queue with attributes that are being inquired upon.

This reason also occurs when the queue is a cluster queue that resolved to a remote instance of the queue. In this case only a subset of the attributes that are valid for local queues can be inquired. See the usage notes in the description of MQINQ in MQINQ - Inquire object attributes for more information about MQINQ.

The call completes with MQCC_WARNING, with the attribute values for the inapplicable selectors set as follows:
- For integer attributes, the corresponding elements of *IntAttrs* are set to MQIAV_NOT_APPLICABLE.
- For character attributes, the appropriate parts of the *CharAttrs* string are set to a character string consisting entirely of asterisks (*).

Completion Code

MQCC_WARNING

Programmer response

Verify that the selector specified is the one that was intended.

If the queue is a cluster queue, specifying one of the MQOO_BROWSE, MQOO_INPUT_*, or MQOO_SET options in addition to MQOO_INQUIRE forces the queue to resolve to the local instance of the queue. However, if there is no local instance of the queue the MQOPEN call fails.

2069 (0815) (RC2069): MQRC_SIGNAL_OUTSTANDING:

Explanation

An MQGET call was issued with either the MQGMO_SET_SIGNAL or MQGMO_WAIT option, but there is already a signal outstanding for the queue handle *Hobj*.

This reason code occurs only in the following environments: z/OS, Windows 95, Windows 98.

Completion Code

MQCC_FAILED

Programmer response

Check the application logic. If it is necessary to set a signal or wait when there is a signal outstanding for the same queue, a different object handle must be used.

2070 (0816) (RC2070): MQRC_SIGNAL_REQUEST_ACCEPTED:

Explanation

An MQGET call was issued specifying MQGMO_SET_SIGNAL in the *GetMsgOpts* parameter, but no suitable message was available; the call returns immediately. The application can now wait for the signal to be delivered.

- On z/OS, the application should wait on the Event Control Block pointed to by the *Signal1* field.
- On Windows 95, Windows 98, the application should wait for the signal Windows message to be delivered.

This reason code occurs only in the following environments: z/OS, Windows 95, Windows 98.

Completion Code

MQCC_WARNING

Programmer response

Wait for the signal; when it is delivered, check the signal to ensure that a message is now available. If it is, reissue the MQGET call.

- On z/OS, wait on the ECB pointed to by the *Signal1* field and, when it is posted, check it to ensure that a message is now available.
- On Windows 95, Windows 98, the application (thread) should continue executing its message loop.

2071 (0817) (RC2071): MQRC_STORAGE_NOT_AVAILABLE:

Explanation

The call failed because there is insufficient main storage available.

Completion Code

MQCC_FAILED

Programmer response

Ensure that active applications are behaving correctly, for example, that they are not looping unexpectedly. If no problems are found, make more main storage available.

- On z/OS, if no application problems are found, ask your system programmer to increase the size of the region in which the queue manager runs.

2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE:

Explanation

Either the MQGMO_SYNCPOINT option was used with an MQGET call, or the MQPMO_SYNCPOINT option was used with an MQPUT or MQPUT1 call, but the local queue manager was unable to honor the request. If the queue manager does not support units of work, the *SyncPoint* queue-manager attribute has the value MQSP_NOT_AVAILABLE.

This reason code can also occur on the MQGET, MQPUT, and MQPUT1 calls when an external unit-of-work coordinator is used. If that coordinator requires an explicit call to start the unit of work, but the application has not issued that call before the MQGET, MQPUT, or MQPUT1 call, reason code MQRC_SYNCPOINT_NOT_AVAILABLE is returned.

- IBM i On IBM i, this reason code means that IBM i Commitment Control is not started, or is unavailable for use by the queue manager.
- On HP Integrity NonStop Server, this reason code means that the client has detected that the application has an active transaction that is being coordinated by the Transaction Management Facility (TMF), but that a z/OS queue manager is unable to be coordinated by TMF.

This reason code can also be returned if the MQGMO_SYNCPOINT or the MQPMO_SYNCPOINT option was used for IBM MQ Multicast messaging. Transactions are not supported for multicast.

Completion Code

MQCC_FAILED

Programmer response

Remove the specification of MQGMO_SYNCPOINT or MQPMO_SYNCPOINT, as appropriate.

- IBM i On IBM i, ensure that Commitment Control is started. If this reason code occurs after Commitment Control is started, contact your system programmer.
- On HP Integrity NonStop Server, ensure that your z/OS queue manager has the relevant APAR applied. Check with the IBM support center for APAR details.

2075 (081B) (RC2075): MQRC_TRIGGER_CONTROL_ERROR:

Explanation

On an MQSET call, the value specified for the MQIA_TRIGGER_CONTROL attribute selector is not valid.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value.

2076 (081C) (RC2076): MQRC_TRIGGER_DEPTH_ERROR:

Explanation

On an MQSET call, the value specified for the MQIA_TRIGGER_DEPTH attribute selector is not valid.

Completion Code

MQCC_FAILED

Programmer response

Specify a value that is greater than zero.

2077 (081D) (RC2077): MQRC_TRIGGER_MSG_PRIORITY_ERR:

Explanation

On an MQSET call, the value specified for the MQIA_TRIGGER_MSG_PRIORITY attribute selector is not valid.

Completion Code

MQCC_FAILED

Programmer response

Specify a value in the range zero through the value of *MaxPriority* queue-manager attribute.

2078 (081E) (RC2078): MQRC_TRIGGER_TYPE_ERROR:

Explanation

On an MQSET call, the value specified for the MQIA_TRIGGER_TYPE attribute selector is not valid.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value.

2079 (081F) (RC2079): MQRC_TRUNCATED_MSG_ACCEPTED:

Explanation

On an MQGET call, the message length was too large to fit into the supplied buffer. The MQGMO_ACCEPT_TRUNCATED_MSG option was specified, so the call completes. The message is removed from the queue (subject to unit-of-work considerations), or, if this was a browse operation, the browse cursor is advanced to this message.

The `DataLength` parameter is set to the length of the message before truncation, the `Buffer` parameter contains as much of the message as fits, and the MQMD structure is filled in.

Completion Code

MQCC_WARNING

Programmer response

None, because the application expected this situation.

2080 (0820) (RC2080): MQRC_TRUNCATED_MSG_FAILED:

Explanation

On an MQGET call, the message length was too large to fit into the supplied buffer. The MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, so the message has not been removed from the queue. If this was a browse operation, the browse cursor remains where it was before this call, but if MQGMO_BROWSE_FIRST was specified, the browse cursor is positioned logically before the highest-priority message on the queue.

The `DataLength` field is set to the length of the message before truncation, the `Buffer` parameter contains as much of the message as fits, and the MQMD structure is filled in.

Completion Code

MQCC_WARNING

Programmer response

Supply a buffer that is at least as large as `DataLength`, or specify MQGMO_ACCEPT_TRUNCATED_MSG if not all of the message data is required.

2082 (0822) (RC2082): MQRC_UNKNOWN_ALIAS_BASE_Q:

Explanation

An MQOPEN or MQPUT1 call was issued specifying an alias queue as the target, but the *BaseQName* in the alias queue attributes is not recognized as a queue name.

This reason code can also occur when *BaseQName* is the name of a cluster queue that cannot be resolved successfully.

MQRC_UNKNOWN_ALIAS_BASE_Q might indicate that the application is specifying the **ObjectQmgrName** of the queue manager that it is connecting to, and the queue manager that is hosting the alias queue. This means that the queue manager looks for the alias target queue on the specified queue manager and fails because the alias target queue is not on the local queue manager. Leave the **ObjectQmgrName** parameter blank so that the clustering decides which queue manager to route to.

Completion Code

MQCC_FAILED

Programmer response

Correct the queue definitions.

This reason code is also used to identify the corresponding event message Unknown Alias Base Queue.

If the reason code is seen by an application that is using IBM MQ classes for JMS, modify the JMS queue object definition that is used by the application so that the **QMANAGER** property is set to the empty string (""). This setting ensures that the clustering decides which queue manager to route to.

2085 (0825) (RC2085): MQRC_UNKNOWN_OBJECT_NAME:

Explanation

An MQOPEN, MQPUT1 , or MQSUB call was issued, but the object identified by the *ObjectName* and *ObjectQMgrName* fields in the object descriptor MQOD cannot be found. One of the following applies:

- The *ObjectQMgrName* field is one of the following:
  - Blank
  - The name of the local queue manager
  - The name of a local definition of a remote queue (a queue-manager alias) in which the *RemoteQMgrName* attribute is the name of the local queue manager

  but no object with the specified *ObjectName* and *ObjectType* exists on the local queue manager.
- The object being opened is a cluster queue that is hosted on a remote queue manager, but the local queue manager does not have a defined route to the remote queue manager.
- The object being opened is a queue definition that has QSGDISP(GROUP). Such definitions cannot be used with the MQOPEN, MQPUT1 , or MQSUB calls.
- The MQOD in the failing application specifies the name of the local queue manager in *ObjectQMgrName*. The local queue manager does not host the particular cluster queue specified in *ObjectName*.

  The solution in this environment is to leave *ObjectQMgrName* of the MQOD blank.

This can also occur in response to a command that specifies the name of an object or other item that does not exist.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid object name. Ensure that the name is padded with blanks at the end, if necessary. If this is correct, check the object definitions.

This reason code is also used to identify the corresponding event message Unknown Object Name.

2086 (0826) (RC2086): MQRC_UNKNOWN_OBJECT_Q_MGR:

Explanation

On an MQOPEN or MQPUT1 call, the *ObjectQMgrName* field in the object descriptor MQOD does not satisfy the naming rules for objects. For more information, see ObjectQMgrName (MQCHAR48).

This reason also occurs if the *ObjectType* field in the object descriptor has the value MQOT_Q_MGR, and the *ObjectQMgrName* field is not blank, but the name specified is not the name of the local queue manager.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid queue manager name. To refer to the local queue manager, a name consisting entirely of blanks or beginning with a null character can be used. Ensure that the name is padded with blanks at the end, or terminated with a null character if necessary.

2087 (0827) (RC2087): MQRC_UNKNOWN_REMOTE_Q_MGR:

Explanation

On an MQOPEN or MQPUT1 call, an error occurred with the queue-name resolution, for one of the following reasons:

- *ObjectQMgrName* is blank or the name of the local queue manager, *ObjectName* is the name of a local definition of a remote queue (or an alias to one), and one of the following is true:
  - *RemoteQMgrName* is blank or the name of the local queue manager. Note that this error occurs even if *XmitQName* is not blank.
  - *XmitQName* is blank, but there is no transmission queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
  - *RemoteQMgrName* and *RemoteQName* specify a cluster queue that cannot be resolved successfully, and the *DefXmitQName* queue-manager attribute is blank.
  - On z/OS only, the *RemoteQMgrName* is the name of a queue manager in the Queue Sharing group but intra-group queuing is disabled.
- *ObjectQMgrName* is the name of a local definition of a remote queue (containing a queue-manager alias definition), and one of the following is true:
  - *RemoteQName* is not blank.
  - *XmitQName* is blank, but there is no transmission queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
- *ObjectQMgrName* is not:
  - Blank

  - The name of the local queue manager
  - The name of a transmission queue
  - The name of a queue-manager alias definition (that is, a local definition of a remote queue with a blank *RemoteQName*)

  but the *DefXmitQName* queue-manager attribute is blank and the queue manager is not part of a queue-sharing group with intra-group queuing enabled.
- *ObjectQMgrName* is the name of a model queue.
- The queue name is resolved through a cell directory. However, there is no queue defined with the same name as the remote queue manager name obtained from the cell directory, and the *DefXmitQName* queue-manager attribute is blank.

Completion Code

MQCC_FAILED

Programmer response

Check the values specified for *ObjectQMgrName* and *ObjectName*. If these are correct, check the queue definitions.

This reason code is also used to identify the corresponding event message Unknown Remote Queue Manager.

2090 (082A) (RC2090): MQRC_WAIT_INTERVAL_ERROR:

Explanation

On the MQGET call, the value specified for the *WaitInterval* field in the *GetMsgOpts* parameter is not valid.

Completion Code

MQCC_FAILED

Programmer response

Specify a value greater than or equal to zero, or the special value MQWI_UNLIMITED if an indefinite wait is required.

2091 (082B) (RC2091): MQRC_XMIT_Q_TYPE_ERROR:

Explanation

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition:

- *XmitQName* is not blank, but specifies a queue that is not a local queue
- *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that is not a local queue

This reason also occurs if the queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a queue, but this is not a local queue.

Completion Code

MQCC_FAILED

Programmer response

Check the values specified for *ObjectName* and *ObjectQMgrName*. If these are correct, check the queue definitions.

This reason code is also used to identify the corresponding event message Transmission Queue Type Error.

2092 (082C) (RC2092): MQRC_XMIT_Q_USAGE_ERROR:

Explanation

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager, but one of the following occurred:

- *ObjectQMgrName* specifies the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.
- The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition:
  - *XmitQName* is not blank, but specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION
  - *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION
  - *XmitQName* specifies the queue SYSTEM.QSG.TRANSMIT.QUEUE the IGQ queue manager attribute indicates that IGQ is DISABLED.
- The queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.

Completion Code

MQCC_FAILED

Programmer response

Check the values specified for *ObjectName* and *ObjectQMgrName*. If these are correct, check the queue definitions.

This reason code is also used to identify the corresponding event message Transmission Queue Usage Error.

2093 (082D) (RC2093): MQRC_NOT_OPEN_FOR_PASS_ALL:

Explanation

An MQPUT call was issued with the MQPMO_PASS_ALL_CONTEXT option specified in the *PutMsgOpts* parameter, but the queue had not been opened with the MQOO_PASS_ALL_CONTEXT option.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_PASS_ALL_CONTEXT (or another option that implies it) when the queue is opened.

2094 (082E) (RC2094): MQRC_NOT_OPEN_FOR_PASS_IDENT:

Explanation

An MQPUT call was issued with the MQPMO_PASS_IDENTITY_CONTEXT option specified in the *PutMsgOpts* parameter, but the queue had not been opened with the MQOO_PASS_IDENTITY_CONTEXT option.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_PASS_IDENTITY_CONTEXT (or another option that implies it) when the queue is opened.

2095 (082F) (RC2095): MQRC_NOT_OPEN_FOR_SET_ALL:

Explanation

An MQPUT call was issued with the MQPMO_SET_ALL_CONTEXT option specified in the *PutMsgOpts* parameter, but the queue had not been opened with the MQOO_SET_ALL_CONTEXT option.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_SET_ALL_CONTEXT when the queue is opened.

2096 (0830) (RC2096): MQRC_NOT_OPEN_FOR_SET_IDENT:

Explanation

An MQPUT call was issued with the MQPMO_SET_IDENTITY_CONTEXT option specified in the `PutMsgOpts` parameter, but the queue had not been opened with the MQOO_SET_IDENTITY_CONTEXT option.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_SET_IDENTITY_CONTEXT (or another option that implies it) when the queue is opened.

2097 (0831) (RC2097): MQRC_CONTEXT_HANDLE_ERROR:

Explanation

On an MQPUT or MQPUT1 call, MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT was specified, but the handle specified in the `Context` field of the `PutMsgOpts` parameter is either not a valid queue handle, or it is a valid queue handle but the queue was not opened with MQOO_SAVE_ALL_CONTEXT.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_SAVE_ALL_CONTEXT when the queue referred to is opened.

2098 (0832) (RC2098): MQRC_CONTEXT_NOT_AVAILABLE:

Explanation

On an MQPUT or MQPUT1 call, MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT was specified, but the queue handle specified in the `Context` field of the `PutMsgOpts` parameter has no context associated with it. This arises if no message has yet been successfully retrieved with the queue handle referred to, or if the last successful MQGET call was a browse.

This condition does not arise if the message that was last retrieved had no context associated with it.

- On z/OS, if a message is received by a message channel agent that is putting messages with the authority of the user identifier in the message, this code is returned in the `Feedback` field of an exception report if the message has no context associated with it.

Completion Code

MQCC_FAILED

Programmer response

Ensure that a successful nonbrowse get call has been issued with the queue handle referred to.

2099 (0833) (RC2099): MQRC_SIGNAL1_ERROR:

Explanation

An MQGET call was issued, specifying MQGMO_SET_SIGNAL in the *GetMsgOpts* parameter, but the *Signal1* field is not valid.

- On z/OS, the address contained in the *Signal1* field is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- On Windows 95, Windows 98, the window handle in the *Signal1* field is not valid.

This reason code occurs only in the following environments: z/OS, Windows 95, Windows 98.

Completion Code

MQCC_FAILED

Programmer response

Correct the setting of the *Signal1* field.

2100 (0834) (RC2100): MQRC_OBJECT_ALREADY_EXISTS:

Explanation

An MQOPEN call was issued to create a dynamic queue, but a queue with the same name as the dynamic queue already exists.

- On z/OS, a rare "race condition⌂ can also give rise to this reason code; see the description of reason code MQRC_NAME_IN_USE for more details.

Completion Code

MQCC_FAILED

Programmer response

If supplying a dynamic queue name in full, ensure that it obeys the naming conventions for dynamic queues; if it does, either supply a different name, or delete the existing queue if it is no longer required. Alternatively, allow the queue manager to generate the name.

If the queue manager is generating the name (either in part or in full), reissue the MQOPEN call.

2101 (0835) (RC2101): MQRC_OBJECT_DAMAGED:

Explanation

The object accessed by the call is damaged and cannot be used. For example, this might be because the definition of the object in main storage is not consistent, or because it differs from the definition of the object on disk, or because the definition on disk cannot be read. The object can be deleted, although it might not be possible to delete the associated user space.

- On z/OS, this reason occurs when the Db2 list header or structure number associated with a shared queue is zero. This situation arises as a result of using the MQSC command DELETE CFSTRUCT to delete the Db2 structure definition. The command resets the list header and structure number to zero for each of the shared queues that references the deleted CF structure.

Completion Code

MQCC_FAILED

Programmer response

It might be necessary to stop and restart the queue manager, or to restore the queue-manager data from backup storage.

- On IBM i, HP Integrity NonStop Server, and UNIX systems, consult the FFST record to obtain more detail about the problem.
- On z/OS, delete the shared queue and redefine it using the MQSC command DEFINE QLOCAL. This automatically defines a CF structure and allocates list headers for it.

2102 (0836) (RC2102): MQRC_RESOURCE_PROBLEM:

Explanation

There are insufficient system resources to complete the call successfully. On z/OS this can indicate that Db2 errors occurred when using shared queues, or that the maximum number of shared queues that can be defined in a single coupling facility list structure has been reached.

Completion Code

MQCC_FAILED

Programmer response

Run the application when the machine is less heavily loaded.

- On z/OS, check the operator console for messages that might provide additional information.
- On IBM i, HP Integrity NonStop Server, and UNIX systems, consult the FFST record to obtain more detail about the problem.

2103 (0837) (RC2103): MQRC_ANOTHER_Q_MGR_CONNECTED:
Explanation

An MQCONN or MQCONNX call was issued, but the thread or process is already connected to a different queue manager. The thread or process can connect to only one queue manager at a time.
- On z/OS, this reason code does not occur.
- On Windows, MTS objects do not receive this reason code, as connections to other queue managers are allowed.

Completion Code

MQCC_FAILED

Programmer response

Use the MQDISC call to disconnect from the queue manager that is already connected, and then issue the MQCONN or MQCONNX call to connect to the new queue manager.

Disconnecting from the existing queue manager closes any queues that are currently open; it is suggested that any uncommitted units of work are committed or backed out before the MQDISC call is issued.

2104 (0838) (RC2104): MQRC_UNKNOWN_REPORT_OPTION:

Explanation

An MQPUT or MQPUT1 call was issued, but the *Report* field in the message descriptor MQMD contains one or more options that are not recognized by the local queue manager. The options are accepted.

The options that cause this reason code to be returned depend on the destination of the message; see the description of REPORT in Report options and message flags for more information.

Completion Code

MQCC_WARNING

Programmer response

If this reason code is expected, no corrective action is required. If this reason code is not expected, do the following:
- Ensure that the *Report* field in the message descriptor is initialized with a value when the message descriptor is declared, or is assigned a value prior to the MQPUT or MQPUT1 call.
- Ensure that the report options specified are valid; see the *Report* field described in the description of MQMD in MQMD - Message descriptor for valid report options.
- If multiple report options are being set by adding the individual report options together, ensure that the same report option is not added twice.
- Check that conflicting report options are not specified. For example, do not add both MQRO_EXCEPTION and MQRO_EXCEPTION_WITH_DATA to the *Report* field; only one of these can be specified.

2105 (0839) (RC2105): MQRC_STORAGE_CLASS_ERROR:

Explanation

The MQPUT or MQPUT1 call was issued, but the storage-class object defined for the queue does not exist.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Create the storage-class object required by the queue, or modify the queue definition to use an existing storage class. The name of the storage-class object used by the queue is given by the `StorageClass` queue attribute.

2106 (083A) (RC2106): MQRC_COD_NOT_VALID_FOR_XCF_Q:

Explanation

An MQPUT or MQPUT1 call was issued, but the `Report` field in the message descriptor MQMD specifies one of the MQRO_COD_* options and the target queue is an XCF queue. MQRO_COD_* options cannot be specified for XCF queues.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Remove the relevant MQRO_COD_* option.

2107 (083B) (RC2107): MQRC_XWAIT_CANCELED:

Explanation

An MQXWAIT call was issued, but the call has been canceled because a STOP CHINIT command has been issued (or the queue manager has been stopped, which causes the same effect). See MQXWAIT for more information about the MQXWAIT call.

Completion Code

MQCC_FAILED

Programmer response

Tidy up and terminate.

2108 (083C) (RC2108): MQRC_XWAIT_ERROR:

Explanation

An MQXWAIT call was issued, but the invocation was not valid for one of the following reasons:
- The wait descriptor MQXWD contains data that is not valid.
- The linkage stack level is not valid.
- The addressing mode is not valid.
- There are too many wait events outstanding.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Obey the rules for using the MQXWAIT call. For more information about MQWAIT, see MQXWAIT.

2109 (083D) (RC2109): MQRC_SUPPRESSED_BY_EXIT:

Explanation

On any call other than MQCONN or MQDISC, the API crossing exit suppressed the call.

Completion Code

MQCC_FAILED

Programmer response

Obey the rules for MQI calls that the exit enforces. To find out the rules, see the writer of the exit.

2110 (083E) (RC2110): MQRC_FORMAT_ERROR:

Explanation

An MQGET call was issued with the MQGMO_CONVERT option specified in the *GetMsgOpts* parameter, but the message cannot be converted successfully due to an error associated with the message format. Possible errors include:
- The format name in the message is MQFMT_NONE.
- A user-written exit with the name specified by the *Format* field in the message cannot be found.
- The message contains data that is not consistent with the format definition.

The message is returned unconverted to the application issuing the MQGET call, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

Completion Code

MQCC_WARNING

Programmer response

Check the format name that was specified when the message was put. If this is not one of the built-in formats, check that a suitable exit with the same name as the format is available for the queue manager to load. Verify that the data in the message corresponds to the format expected by the exit.

2111 (083F) (RC2111): MQRC_SOURCE_CCSID_ERROR:

Explanation

The coded character-set identifier from which character data is to be converted is not valid or not supported.

This can occur on the MQGET call when the MQGMO_CONVERT option is included in the *GetMsgOpts* parameter; the coded character-set identifier in error is the *CodedCharSetId* field in the message being retrieved. In this case, the message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

This reason can also occur on the MQGET call when the message contains one or more MQ header structures (MQCIH, MQDLH, MQIIH, MQRMH), and the *CodedCharSetId* field in the message specifies a character set that does not have SBCS characters for the characters that are valid in queue names. MQ header structures containing such characters are not valid, and so the message is returned unconverted. The Unicode character set UCS-2 is an example of such a character set.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

This reason can also occur on the MQXCNVC call; the coded character-set identifier in error is the *SourceCCSID* parameter. Either the *SourceCCSID* parameter specifies a value that is not valid or not supported, or the *SourceCCSID* parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason can also occur on a MQSETMP/MQINQMP/MQDLTMP call when the application issuing the calls does not use Language Environment (LE) and defines CCSID values of MQCCSI_APPL (-3) for message property names and string property values.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Check the character-set identifier that was specified when the message was put, or that was specified for the *SourceCCSID* parameter on the MQXCNVC call. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the specified character set, conversion must be carried out by the application.

If this reason happens as result of a MQSETMP/MQINQMP/MQDLTMP call issued in a non-LE application program that has specified CCSID as MQCCSI_APPL (-3) then applications should be changed to specify the CCSID value used by the application to encode the property names or property string values.

Your applications should override the value of MQCCSI_APPL (-3) with the correct CCSID used as described in Redefinition of MQCCSI_APPL, or they should set the explicit CCSID value used to encode text strings in MQCHARV or similar structures.

2112 (0840) (RC2112): MQRC_SOURCE_INTEGER_ENC_ERROR:

Explanation

On an MQGET call, with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the message being retrieved specifies an integer encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

This reason code can also occur on the MQXCNVC call, when the *Options* parameter contains an unsupported MQDCC_SOURCE_* value, or when MQDCC_SOURCE_ENC_UNDEFINED is specified for a UCS-2 code page.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Check the integer encoding that was specified when the message was put. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required integer encoding, conversion must be carried out by the application.

2113 (0841) (RC2113): MQRC_SOURCE_DECIMAL_ENC_ERROR:

Explanation

On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the message being retrieved specifies a decimal encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

Completion Code

MQCC_WARNING

Programmer response

Check the decimal encoding that was specified when the message was put. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required decimal encoding, conversion must be carried out by the application.

2114 (0842) (RC2114): MQRC_SOURCE_FLOAT_ENC_ERROR:

Explanation

On an MQGET call, with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the message being retrieved specifies a floating-point encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

Completion Code

MQCC_WARNING

Programmer response

Check the floating-point encoding that was specified when the message was put. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required floating-point encoding, conversion must be carried out by the application.

2115 (0843) (RC2115): MQRC_TARGET_CCSID_ERROR:

Explanation

The coded character-set identifier to which character data is to be converted is not valid or not supported.

This can occur on the MQGET call when the MQGMO_CONVERT option is included in the *GetMsgOpts* parameter; the coded character-set identifier in error is the *CodedCharSetId* field in the *MsgDesc* parameter. In this case, the message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

This reason can also occur on the MQGET call when the message contains one or more MQ header structures (MQCIH, MQDLH, MQIIH, MQRMH), and the *CodedCharSetId* field in the *MsgDesc* parameter specifies a character set that does not have SBCS characters for the characters that are valid in queue names. The Unicode character set UCS-2 is an example of such a character set.

This reason can also occur on the MQXCNVC call; the coded character-set identifier in error is the *TargetCCSID* parameter. Either the *TargetCCSID* parameter specifies a value that is not valid or not supported, or the *TargetCCSID* parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Check the character-set identifier that was specified for the *CodedCharSetId* field in the *MsgDesc* parameter on the MQGET call, or that was specified for the *SourceCCSID* parameter on the MQXCNVC call. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the specified character set, conversion must be carried out by the application.

2116 (0844) (RC2116): MQRC_TARGET_INTEGER_ENC_ERROR:

Explanation

On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the *MsgDesc* parameter specifies an integer encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message being retrieved, and the call completes with MQCC_WARNING.

This reason code can also occur on the MQXCNVC call, when the *Options* parameter contains an unsupported MQDCC_TARGET_* value, or when MQDCC_TARGET_ENC_UNDEFINED is specified for a UCS-2 code page.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Check the integer encoding that was specified. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required integer encoding, conversion must be carried out by the application.

2117 (0845) (RC2117): MQRC_TARGET_DECIMAL_ENC_ERROR:

Explanation

On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the *MsgDesc* parameter specifies a decimal encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

Completion Code

MQCC_WARNING

Programmer response

Check the decimal encoding that was specified. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required decimal encoding, conversion must be carried out by the application.

2118 (0846) (RC2118): MQRC_TARGET_FLOAT_ENC_ERROR:

Explanation

On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the *MsgDesc* parameter specifies a floating-point encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

Completion Code

MQCC_WARNING

Programmer response

Check the floating-point encoding that was specified. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required floating-point encoding, conversion must be carried out by the application.

2119 (0847) (RC2119): MQRC_NOT_CONVERTED:

Explanation

An MQGET call was issued with the MQGMO_CONVERT option specified in the *GetMsgOpts* parameter, but an error occurred during conversion of the data in the message. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

This error may also indicate that a parameter to the data-conversion service is not supported.

Completion Code

MQCC_WARNING

Programmer response

Check that the message data is correctly described by the *Format*, *CodedCharSetId* and *Encoding* parameters that were specified when the message was put. Also check that these values, and the *CodedCharSetId* and *Encoding* specified in the *MsgDesc* parameter on the MQGET call, are supported for queue-manager conversion. If the required conversion is not supported, conversion must be carried out by the application.

2120 (0848) (RC2120): MQRC_CONVERTED_MSG_TOO_BIG:

Explanation

On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the message data expanded during data conversion and exceeded the size of the buffer provided by the application. However, the message had already been removed from the queue because prior to conversion the message data could be accommodated in the application buffer without truncation.

The message is returned unconverted, with the *CompCode* parameter of the MQGET call set to MQCC_WARNING. If the message consists of several parts, each of which is described by its own character-set and encoding fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various character-set and encoding fields always correctly describe the relevant message data.

This reason also occurs on the MQXCNVC call, when the *TargetBuffer* parameter is too small to accommodate the converted string, and the string has been truncated to fit in the buffer. The length of valid data returned is given by the *DataLength* parameter; in the case of a DBCS string or mixed SBCS/DBCS string, this length may be *less than* the length of *TargetBuffer*.

Completion Code

MQCC_WARNING

Programmer response

For the MQGET call, check that the exit is converting the message data correctly and setting the output length *DataLength* to the appropriate value. If it is, the application issuing the MQGET call must provide a larger buffer for the *Buffer* parameter.

For the MQXCNVC call, if the string must be converted without truncation, provide a larger output buffer.

2121 (0849) (RC2121): MQRC_NO_EXTERNAL_PARTICIPANTS:

Explanation

An MQBEGIN call was issued to start a unit of work coordinated by the queue manager, but no participating resource managers have been registered with the queue manager. As a result, only changes to MQ resources can be coordinated by the queue manager in the unit of work.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows.

Completion Code

MQCC_WARNING

Programmer response

If the application does not require non-MQ resources to participate in the unit of work, this reason code can be ignored or the MQBEGIN call removed. Otherwise consult your system programmer to determine why the required resource managers have not been registered with the queue manager; the queue manager's configuration file might be in error.

2122 (084A) (RC2122): MQRC_PARTICIPANT_NOT_AVAILABLE:

Explanation

An MQBEGIN call was issued to start a unit of work coordinated by the queue manager, but one or more of the participating resource managers that had been registered with the queue manager is not available. As a result, changes to those resources cannot be coordinated by the queue manager in the unit of work.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows.

Completion Code

MQCC_WARNING

Programmer response

If the application does not require non-MQ resources to participate in the unit of work, this reason code can be ignored. Otherwise consult your system programmer to determine why the required resource managers are not available. The resource manager might have been halted temporarily, or there might be an error in the queue manager's configuration file.

2123 (084B) (RC2123): MQRC_OUTCOME_MIXED:

Explanation

The queue manager is acting as the unit-of-work coordinator for a unit of work that involves other resource managers, but one of the following occurred:
- An MQCMIT or MQDISC call was issued to commit the unit of work, but one or more of the participating resource managers backed-out the unit of work instead of committing it. As a result, the outcome of the unit of work is mixed.
- An MQBACK call was issued to back out a unit of work, but one or more of the participating resource managers had already committed the unit of work.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Examine the queue-manager error logs for messages relating to the mixed outcome; these messages identify the resource managers that are affected. Use procedures local to the affected resource managers to resynchronize the resources.

This reason code does not prevent the application initiating further units of work.

2124 (084C) (RC2124): MQRC_OUTCOME_PENDING:

Explanation

The queue manager is acting as the unit-of-work coordinator for a unit of work that involves other resource managers, and an MQCMIT or MQDISC call was issued to commit the unit of work, but one or more of the participating resource managers has not confirmed that the unit of work was committed successfully.

The completion of the commit operation will happen at some point in the future, but there remains the possibility that the outcome will be mixed.

AIX HP-UX Solaris Windows This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

z/OS On z/OS, this situation can occur if a queue manager loses connectivity to a coupling facility structure while a unit of work that affects messages on shared queues is being committed or backed out.

Completion Code

MQCC_WARNING

Programmer response

AIX HP-UX Solaris Windows Use the normal error-reporting mechanisms to determine whether the outcome was mixed. If it was, take appropriate action to resynchronize the resources.

AIX HP-UX Solaris Windows This reason code does not prevent the application initiating further units of work.

z/OS If this reason code was returned as a result of loss of connectivity to a coupling facility structure on z/OS, the operation will be completed either when the queue manager reconnects to the affected structure, or when another queue manager in the queue-sharing group is able to perform peer recovery on the structure.

2125 (084D) (RC2125): MQRC_BRIDGE_STARTED:

Explanation

The IMS bridge has been started.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Bridge Started.

2126 (084E) (RC2126): MQRC_BRIDGE_STOPPED:

Explanation

The IMS bridge has been stopped.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Bridge Stopped.

2127 (084F) (RC2127): MQRC_ADAPTER_STORAGE_SHORTAGE:

Explanation

On an MQCONN call, the adapter was unable to acquire storage.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Notify the system programmer. The system programmer should determine why the system is short on storage, and take appropriate action, for example, increase the region size on the step or job card.

2128 (0850) (RC2128): MQRC_UOW_IN_PROGRESS:

Explanation

An MQBEGIN call was issued to start a unit of work coordinated by the queue manager, but a unit of work is already in existence for the connection handle specified. This may be a global unit of work started by a previous MQBEGIN call, or a unit of work that is local to the queue manager or one of the cooperating resource managers. No more than one unit of work can exist concurrently for a connection handle.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Review the application logic to determine why there is a unit of work already in existence. Move the MQBEGIN call to the appropriate place in the application.

2129 (0851) (RC2129): MQRC_ADAPTER_CONN_LOAD_ERROR:

Explanation

On an MQCONN call, the connection handling module could not be loaded, so the adapter could not link to it. The connection handling module name is:

- CSQBCON for batch applications
- CSQQCONN or CSQQCON2 for IMS applications

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct library concatenation has been specified in the batch application program execution JCL, and in the queue-manager startup JCL.

2130 (0852) (RC2130): MQRC_ADAPTER_SERV_LOAD_ERROR:

Explanation

On an MQI call, the batch adapter could not load one of the following API service module, and so could not link to it:

- CSQBSRV
- CSQAPEPL
- CSQBCRMH
- CSQBAPPL

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct library concatenation has been specified in the batch application program execution JCL, and in the queue-manager startup JCL.

2131 (0853) (RC2131): MQRC_ADAPTER_DEFS_ERROR:

Explanation

On an MQCONN call, the subsystem definition module (CSQBDEFV for batch and CSQQDEFV for IMS) does not contain the required control block identifier.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Check your library concatenation. If this is correct, check that the CSQBDEFV or CSQQDEFV module contains the required subsystem ID.

2132 (0854) (RC2132): MQRC_ADAPTER_DEFS_LOAD_ERROR:

Explanation

On an MQCONN call, the subsystem definition module (CSQBDEFV for batch and CSQQDEFV for IMS) could not be loaded.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct library concatenation has been specified in the application program execution JCL, and in the queue-manager startup JCL.

2133 (0855) (RC2133): MQRC_ADAPTER_CONV_LOAD_ERROR:

Explanation

On an MQGET call, the adapter (batch or IMS) could not load the data conversion services modules.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct library concatenation has been specified in the batch application program execution JCL, and in the queue-manager startup JCL.

2134 (0856) (RC2134): MQRC_BO_ERROR:

Explanation

On an MQBEGIN call, the begin-options structure MQBO is not valid, for one of the following reasons:
- The *StrucId* field is not MQBO_STRUC_ID.
- The *Version* field is not MQBO_VERSION_1.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that input fields in the MQBO structure are set correctly.

2135 (0857) (RC2135): MQRC_DH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQDH structure that is not valid. Possible errors include the following:
- The *StrucId* field is not MQDH_STRUC_ID.
- The *Version* field is not MQDH_VERSION_1.
- The *StrucLength* field specifies a value that is too small to include the structure plus the arrays of MQOR and MQPMR records.
- The *CodedCharSetId* field is zero, or a negative value that is not valid.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the *CodedCharSetId* field to a valid value (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in this field).

2136 (0858) (RC2136): MQRC_MULTIPLE_REASONS:

Explanation

An MQOPEN, MQPUT or MQPUT1 call was issued to open a distribution list or put a message to a distribution list, but the result of the call was not the same for all of the destinations in the list. One of the following applies:

- The call succeeded for some of the destinations but not others. The completion code is MQCC_WARNING in this case.
- The call failed for all of the destinations, but for differing reasons. The completion code is MQCC_FAILED in this case.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Examine the MQRR response records to identify the destinations for which the call failed, and the reason for the failure. Ensure that sufficient response records are provided by the application on the call to enable the error(s) to be determined. For the MQPUT1 call, the response records must be specified using the MQOD structure, and not the MQPMO structure.

2137 (0859) (RC2137): MQRC_OPEN_FAILED:

Explanation

A queue or other MQ object could not be opened successfully, for one of the following reasons:

- An MQCONN or MQCONNX call was issued, but the queue manager was unable to open an object that is used internally by the queue manager. As a result, processing cannot continue. The error log will contain the name of the object that could not be opened.
- An MQPUT call was issued to put a message to a distribution list, but the message could not be sent to the destination to which this reason code applies because that destination was not opened successfully by the MQOPEN call. This reason occurs only in the *Reason* field of the MQRR response record.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Do one of the following:

- If the error occurred on the MQCONN or MQCONNX call, ensure that the required objects exist by running the following command and then retrying the application:
  ```
  STRMQM -c qmgr
  ```
  where `qmgr` should be replaced by the name of the queue manager.

- If the error occurred on the MQPUT call, examine the MQRR response records specified on the MQOPEN call to determine the reason that the queue failed to open. Ensure that sufficient response records are provided by the application on the call to enable the error(s) to be determined.

2138 (085A) (RC2138): MQRC_ADAPTER_DISC_LOAD_ERROR:

Explanation

On an MQDISC call, the disconnect handling module (CSQBDSC for batch and CSQQDISC for IMS) could not be loaded, so the adapter could not link to it.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct library concatenation has been specified in the application program execution JCL, and in the queue-manager startup JCL. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

2139 (085B) (RC2139): MQRC_CNO_ERROR:

Explanation

On an MQCONNX call, the connect-options structure MQCNO is not valid, for one of the following reasons:
- The *StrucId* field is not MQCNO_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the parameter pointer points to read-only storage.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that input fields in the MQCNO structure are set correctly.

2140 (085C) (RC2140): MQRC_CICS_WAIT_FAILED:

Explanation

On any MQI call, the CICS adapter issued an EXEC CICS WAIT request, but the request was rejected by CICS.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Examine the CICS trace data for actual response codes. The most likely cause is that the task has been canceled by the operator or by the system.

2141 (085D) (RC2141): MQRC_DLH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQDLH structure that is not valid. Possible errors include the following:

- The *StrucId* field is not MQDLH_STRUC_ID.
- The *Version* field is not MQDLH_VERSION_1.
- The *CodedCharSetId* field is zero, or a negative value that is not valid.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the *CodedCharSetId* field to a valid value (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in this field).

2142 (085E) (RC2142): MQRC_HEADER_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQ header structure that is not valid. Possible errors include the following:

- The *StrucId* field is not valid.
- The *Version* field is not valid.
- The *StrucLength* field specifies a value that is too small.
- The *CodedCharSetId* field is zero, or a negative value that is not valid.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the *CodedCharSetId* field to a valid value (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in this field).

2143 (085F) (RC2143): MQRC_SOURCE_LENGTH_ERROR:

Explanation

On the MQXCNVC call, the *SourceLength* parameter specifies a length that is less than zero or not consistent with the string's character set or content (for example, the character set is a double-byte character set, but the length is not a multiple of two). This reason also occurs if the *SourceLength* parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_SOURCE_LENGTH_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Specify a length that is zero or greater. If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

2144 (0860) (RC2144): MQRC_TARGET_LENGTH_ERROR:

Explanation

On the MQXCNVC call, the `TargetLength` parameter is not valid for one of the following reasons:
- `TargetLength` is less than zero.
- The `TargetLength` parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The MQDCC_FILL_TARGET_BUFFER option is specified, but the value of `TargetLength` is such that the target buffer cannot be filled completely with valid characters. This can occur when `TargetCCSID` is a pure DBCS character set (such as UCS-2), but `TargetLength` specifies a length that is an odd number of bytes.

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_TARGET_LENGTH_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Specify a length that is zero or greater. If the MQDCC_FILL_TARGET_BUFFER option is specified, and `TargetCCSID` is a pure DBCS character set, ensure that `TargetLength` specifies a length that is a multiple of two.

If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

2145 (0861) (RC2145): MQRC_SOURCE_BUFFER_ERROR:

Explanation

On the MQXCNVC call, the `SourceBuffer` parameter pointer is not valid, or points to storage that cannot be accessed for the entire length specified by `SourceLength`. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_SOURCE_BUFFER_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Specify a valid buffer. If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

2146 (0862) (RC2146): MQRC_TARGET_BUFFER_ERROR:

Explanation

On the MQXCNVC call, the *TargetBuffer* parameter pointer is not valid, or points to read-only storage, or to storage that cannot be accessed for the entire length specified by *TargetLength*. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_TARGET_BUFFER_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Specify a valid buffer. If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

2148 (0864) (RC2148): MQRC_IIH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQIIH structure that is not valid. Possible errors include the following:

- The *StrucId* field is not MQIIH_STRUC_ID.
- The *Version* field is not MQIIH_VERSION_1.
- The *StrucLength* field is not MQIIH_LENGTH_1.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2149 (0865) (RC2149): MQRC_PCF_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued to put a message containing PCF data, but the length of the message does not equal the sum of the lengths of the PCF structures present in the message. This can occur for messages with the following format names:
- MQFMT_ADMIN
- MQFMT_EVENT
- MQFMT_PCF

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the length of the message specified on the MQPUT or MQPUT1 call equals the sum of the lengths of the PCF structures contained within the message data.

2150 (0866) (RC2150): MQRC_DBCS_ERROR:

Explanation

An error was encountered attempting to convert a double-byte character set (DBCS) string. This can occur in the following cases:
- On the MQXCNVC call, when the *SourceCCSID* parameter specifies the coded character-set identifier of a double-byte character set, but the *SourceBuffer* parameter does not contain a valid DBCS string. This may be because the string contains characters that are not valid DBCS characters, or because the string is a mixed SBCS/DBCS string and the shift-out/shift-in characters are not correctly paired. The completion code is MQCC_FAILED in this case.
- On the MQGET call, when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_DBCS_ERROR reason code was returned by an MQXCNVC call issued by the data conversion exit. The completion code is MQCC_WARNING in this case.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Specify a valid string.

If the reason code occurs on the MQGET call, check that the data in the message is valid, and that the logic in the data-conversion exit is correct.

2152 (0868) (RC2152): MQRC_OBJECT_NAME_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the `RecsPresent` field in MQOD is greater than zero), but the `ObjectName` field is neither blank nor the null string.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

If it is intended to open a distribution list, set the `ObjectName` field to blanks or the null string. If it is not intended to open a distribution list, set the `RecsPresent` field to zero.

2153 (0869) (RC2153): MQRC_OBJECT_Q_MGR_NAME_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the `RecsPresent` field in MQOD is greater than zero), but the `ObjectQMgrName` field is neither blank nor the null string.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

If it is intended to open a distribution list, set the `ObjectQMgrName` field to blanks or the null string. If it is not intended to open a distribution list, set the `RecsPresent` field to zero.

2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued, but the call failed for one of the following reasons:
- `RecsPresent` in MQOD is less than zero.
- `ObjectType` in MQOD is not MQOT_Q, and `RecsPresent` is not zero. `RecsPresent` must be zero if the object being opened is not a queue.
- IBM MQ Multicast is being used and `RecsPresent` in MQOD is not set to zero. IBM MQ Multicast does not use distribution lists.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

If it is intended to open a distribution list, set the *ObjectType* field to MQOT_Q and *RecsPresent* to the number of destinations in the list. If it is not intended to open a distribution list, set the *RecsPresent* field to zero.

2155 (086B) (RC2155): MQRC_OBJECT_RECORDS_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *RecsPresent* field in MQOD is greater than zero), but the MQOR object records are not specified correctly. One of the following applies:

- *ObjectRecOffset* is zero and *ObjectRecPtr* is zero or the null pointer.
- *ObjectRecOffset* is not zero and *ObjectRecPtr* is not zero and not the null pointer.
- *ObjectRecPtr* is not a valid pointer.
- *ObjectRecPtr* or *ObjectRecOffset* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that one of *ObjectRecOffset* and *ObjectRecPtr* is zero and the other nonzero. Ensure that the field used points to accessible storage.

2156 (086C) (RC2156): MQRC_RESPONSE_RECORDS_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *RecsPresent* field in MQOD is greater than zero), but the MQRR response records are not specified correctly. One of the following applies:

- *ResponseRecOffset* is not zero and *ResponseRecPtr* is not zero and not the null pointer.
- *ResponseRecPtr* is not a valid pointer.
- *ResponseRecPtr* or *ResponseRecOffset* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that at least one of *ResponseRecOffset* and *ResponseRecPtr* is zero. Ensure that the field used points to accessible storage.

2157 (086D) (RC2157): MQRC_ASID_MISMATCH:

Explanation

On any MQI call, the caller's primary ASID was found to be different from the home ASID.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Correct the application (MQI calls cannot be issued in cross-memory mode). Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

2158 (086E) (RC2158): MQRC_PMO_RECORD_FLAGS_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued to put a message, but the `PutMsgRecFields` field in the MQPMO structure is not valid, for one of the following reasons:

- The field contains flags that are not valid.
- The message is being put to a distribution list, and put message records have been provided (that is, `RecsPresent` is greater than zero, and one of `PutMsgRecOffset` or `PutMsgRecPtr` is nonzero), but `PutMsgRecFields` has the value MQPMRF_NONE.
- MQPMRF_ACCOUNTING_TOKEN is specified without either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that `PutMsgRecFields` is set with the appropriate MQPMRF_* flags to indicate which fields are present in the put message records. If MQPMRF_ACCOUNTING_TOKEN is specified, ensure that either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is also specified. Alternatively, set both `PutMsgRecOffset` and `PutMsgRecPtr` to zero.

2159 (086F) (RC2159): MQRC_PUT_MSG_RECORDS_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued to put a message to a distribution list, but the MQPMR put message records are not specified correctly. One of the following applies:
- *PutMsgRecOffset* is not zero and *PutMsgRecPtr* is not zero and not the null pointer.
- *PutMsgRecPtr* is not a valid pointer.
- *PutMsgRecPtr* or *PutMsgRecOffset* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that at least one of *PutMsgRecOffset* and *PutMsgRecPtr* is zero. Ensure that the field used points to accessible storage.

2160 (0870) (RC2160): MQRC_CONN_ID_IN_USE:

Explanation

On an MQCONN call, the connection identifier assigned by the queue manager to the connection between a CICS or IMS allied address space and the queue manager conflicts with the connection identifier of another connected CICS or IMS system. The connection identifier assigned is as follows:
- For CICS, the applid
- For IMS, the IMSID parameter on the IMSCTRL (sysgen) macro, or the IMSID parameter on the execution parameter (EXEC card in IMS control region JCL)
- For batch, the job name
- For TSO, the user ID

A conflict arises only if there are two CICS systems, two IMS systems, or one each of CICS and IMS, having the same connection identifiers. Batch and TSO connections need not have unique identifiers.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the naming conventions used in different systems that might connect to the queue manager do not conflict.

2161 (0871) (RC2161): MQRC_Q_MGR_QUIESCING:

Explanation

An MQI call was issued, but the call failed because the queue manager is quiescing (preparing to shut down).

When the queue manager is quiescing, the MQOPEN, MQPUT, MQPUT1, and MQGET calls can still complete successfully, but the application can request that they fail by specifying the appropriate option on the call:
- MQOO_FAIL_IF_QUIESCING on MQOPEN
- MQPMO_FAIL_IF_QUIESCING on MQPUT or MQPUT1
- MQGMO_FAIL_IF_QUIESCING on MQGET

Specifying these options enables the application to become aware that the queue manager is preparing to shut down.
- On z/OS:
  - For batch applications, this reason can be returned to applications running in LPARs that do not have a queue manager installed.
  - For CICS applications, this reason can be returned when no connection was established.
- On IBM i for applications running in compatibility mode, this reason can be returned when no connection was established.

Completion Code

MQCC_FAILED

Programmer response

The application should tidy up and end. If the application specified the MQOO_FAIL_IF_QUIESCING, MQPMO_FAIL_IF_QUIESCING, or MQGMO_FAIL_IF_QUIESCING option on the failing call, the relevant option can be removed and the call reissued. By omitting these options, the application can continue working to complete and commit the current unit of work, but the application does not start a new unit of work.

2162 (0872) (RC2162): MQRC_Q_MGR_STOPPING:

Explanation

An MQI call was issued, but the call failed because the queue manager is shutting down. If the call was an MQGET call with the MQGMO_WAIT option, the wait has been canceled. No more MQI calls can be issued.

For MQ MQI client applications, it is possible that the call did complete successfully, even though this reason code is returned with a `CompCode` of MQCC_FAILED.
- On z/OS, the MQRC_CONNECTION_BROKEN reason may be returned instead if, as a result of system scheduling factors, the queue manager shuts down before the call completes.

Completion Code

MQCC_FAILED

Programmer response

The application should tidy up and end. If the application is in the middle of a unit of work coordinated by an external unit-of-work coordinator, the application should issue the appropriate call to back out the unit of work. Any unit of work that is coordinated by the queue manager is backed out automatically.

2163 (0873) (RC2163): MQRC_DUPLICATE_RECOV_COORD:

Explanation

On an MQCONN or MQCONNX call, a recovery coordinator already exists for the connection name specified on the connection call issued by the adapter.

A conflict arises only if there are two CICS systems, two IMS systems, or one each of CICS and IMS, having the same connection identifiers. Batch and TSO connections need not have unique identifiers.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the naming conventions used in different systems that might connect to the queue manager do not conflict.

2173 (087D) (RC2173): MQRC_PMO_ERROR:

Explanation

On an MQPUT or MQPUT1 call, the MQPMO structure is not valid, for one of the following reasons:
- The *StrucId* field is not MQPMO_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer response

Ensure that input fields in the MQPMO structure are set correctly.

2182 (0886) (RC2182): MQRC_API_EXIT_NOT_FOUND:

Explanation

The API crossing exit entry point could not be found.

Completion Code

MQCC_FAILED

Programmer response

Check the entry point name is valid for the library module.

2183 (0887) (RC2183): MQRC_API_EXIT_LOAD_ERROR:

Explanation

The API crossing exit module could not be linked to. If this message is returned when the API crossing exit is called *after* the process has been run, the process itself might have completed correctly.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct library concatenation has been specified, and that the API crossing exit module is executable and correctly named. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

2184 (0888) (RC2184): MQRC_REMOTE_Q_NAME_ERROR:

Explanation

On an MQOPEN or MQPUT1 call, one of the following occurred:
- A local definition of a remote queue (or an alias to one) was specified, but the *RemoteQName* attribute in the remote queue definition is entirely blank. Note that this error occurs even if the *XmitQName* in the definition is not blank.
- The *ObjectQMgrName* field in the object descriptor is not blank and not the name of the local queue manager, but the *ObjectName* field is blank.

Completion Code

MQCC_FAILED

Programmer response

Alter the local definition of the remote queue and supply a valid remote queue name, or supply a nonblank *ObjectName* in the object descriptor, as appropriate.

This reason code is also used to identify the corresponding event message Remote Queue Name Error.

2185 (0889) (RC2185): MQRC_INCONSISTENT_PERSISTENCE:

Explanation

An MQPUT call was issued to put a message in a group or a segment of a logical message, but the value specified or defaulted for the *Persistence* field in MQMD is not consistent with the current group and segment information retained by the queue manager for the queue handle. All messages in a group and all segments in a logical message must have the same value for persistence, that is, all must be persistent, or all must be nonpersistent.

If the current call specifies MQPMO_LOGICAL_ORDER, the call fails. If the current call does not specify MQPMO_LOGICAL_ORDER, but the previous MQPUT call for the queue handle did, the call succeeds with completion code MQCC_WARNING.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Modify the application to ensure that the same value of persistence is used for all messages in the group, or all segments of the logical message.

2186 (088A) (RC2186): MQRC_GMO_ERROR:

Explanation

On an MQGET call, the MQGMO structure is not valid, for one of the following reasons:
- The *StrucId* field is not MQGMO_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer response

Ensure that input fields in the MQGMO structure are set correctly.

2187 (088B) (RC2187): MQRC_CICS_BRIDGE_RESTRICTION:

Explanation

It is not permitted to issue MQI calls from user transactions that are run in an MQ/CICS bridge environment where the bridge exit also issues MQI calls. The MQI call fails. If it occurs in the bridge exit, it results in a transaction abend. If it occurs in the user transaction, it can result in a transaction abend.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

The transaction cannot be run using the MQ/CICS bridge. Refer to the appropriate CICS manual for information about restrictions in the MQ/CICS bridge environment.

2188 (088C) (RC2188): MQRC_STOPPED_BY_CLUSTER_EXIT:

Explanation

An MQOPEN, MQPUT, or MQPUT1 call was issued to open or put a message on a cluster queue, but the cluster workload exit rejected the call.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check the cluster workload exit to ensure that it has been written correctly. Determine why it rejected the call and correct the problem.

2189 (088D) (RC2189): MQRC_CLUSTER_RESOLUTION_ERROR:

Explanation

An MQOPEN, MQPUT, or MQPUT1 call was issued to open or put a message on a cluster queue, but the queue definition could not be resolved correctly because a response was required from the repository manager but none was available.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the repository manager is operating and that the queue and channel definitions are correct.

2190 (088E) (RC2190): MQRC_CONVERTED_STRING_TOO_BIG:

Explanation

On an MQGET call with the MQGMO_CONVERT option included in the `GetMsgOpts` parameter, a string in a fixed-length field in the message expanded during data conversion and exceeded the size of the field. When this happens, the queue manager tries discarding trailing blank characters and characters following the first null character to make the string fit, but in this case there were insufficient characters that could be discarded.

This reason code can also occur for messages with a format name of MQFMT_IMS_VAR_STRING. When this happens, it indicates that the IMS variable string expanded such that its length exceeded the capacity of the 2 byte binary length field contained within the structure of the IMS variable string. (The queue manager never discards trailing blanks in an IMS variable string.)

The message is returned unconverted, with the `CompCode` parameter of the MQGET call set to MQCC_WARNING. If the message consists of several parts, each of which is described by its own character-set and encoding fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts might be converted and other parts not converted. However, the values returned in the various character-set and encoding fields always correctly describe the relevant message data.

This reason code does not occur if the string can be made to fit by discarding trailing blank characters.

Completion Code

MQCC_WARNING

Programmer response

Check that the fields in the message contain the correct values, and that the character-set identifiers specified by the sender and receiver of the message are correct. If they are, the layout of the data in the message must be modified to increase the lengths of the field, or fields so that there is sufficient space to permit the string, or strings to expand when converted.

2191 (088F) (RC2191): MQRC_TMC_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQTMC2 structure that is not valid. Possible errors include the following:
- The `StrucId` field is not MQTMC_STRUC_ID.
- The `Version` field is not MQTMC_VERSION_2.
- The `BufferLength` parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2192 (0890) (RC2192): MQRC_PAGESET_FULL:

Explanation

Former name for MQRC_STORAGE_MEDIUM_FULL.

2192 (0890) (RC2192): MQRC_STORAGE_MEDIUM_FULL:

Explanation

An MQI call or command was issued to operate on an object, but the call failed because the external storage medium is full. One of the following applies:

- A page-set data set is full (nonshared queues only).
- A coupling-facility structure is full (shared queues only).
- A coupling-facility is full. This situation can arise when the coupling facility structure is configured to use SCM storage (SCMMAXSIZE configured in CFRM policy) and messages are offloaded to SCM storage because the coupling facility structure has reached 90% threshold. Additional SCM use requires further augmented storage for the structure and there is insufficient storage in the coupling-facility to support this.
- The SMDS was full.

You can get this reason code when the page set or SMDS were expanding, but the space was not yet available. Check the messages in the job log to see the status of any expansion.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Check which queues contain messages and look for applications that might be filling the queues unintentionally. Be aware that the queue that has caused the page set or coupling-facility structure to become full is not necessarily the queue referenced by the MQI call that returned MQRC_STORAGE_MEDIUM_FULL.

Check that all of the usual server applications are operating correctly and processing the messages on the queues.

If the applications and servers are operating correctly, increase the number of server applications to cope with the message load, or request the system programmer to increase the size of the page-set data sets.

2193 (0891) (RC2193): MQRC_PAGESET_ERROR:

Explanation

An error was encountered with the page set while attempting to access it for a locally defined queue. This could be because the queue is on a page set that does not exist. A console message is issued that tells you the number of the page set in error. For example if the error occurred in the TEST job, and your user identifier is ABCDEFG, the message is:

```
CSQI041I CSQIALLC JOB TEST USER ABCDEFG HAD ERROR ACCESSING PAGE SET 27
```

If this reason code occurs while attempting to delete a dynamic queue with MQCLOSE, the dynamic queue has not been deleted.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Check that the storage class for the queue maps to a valid page set using the DISPLAY Q(xx) STGCLASS, DISPLAY STGCLASS(xx), and DISPLAY USAGE PSID commands. If you are unable to resolve the problem, notify the system programmer who should:

- Collect the following diagnostic information:
  - A description of the actions that led to the error
  - A listing of the application program being run at the time of the error
  - Details of the page sets defined for use by the queue manager
- Attempt to re-create the problem, and take a system dump immediately after the error occurs
- Contact your IBM Support Center

2194 (0892) (RC2194): MQRC_NAME_NOT_VALID_FOR_TYPE:

Explanation

An MQOPEN call was issued to open the queue manager definition, but the *ObjectName* field in the *ObjDesc* parameter is not blank.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the *ObjectName* field is set to blanks.

2195 (0893) (RC2195): MQRC_UNEXPECTED_ERROR:

Explanation

The call was rejected because an unexpected error occurred.

Completion Code

MQCC_FAILED

Programmer response

Check the application's parameter list to ensure, for example, that the correct number of parameters was passed, and that data pointers and storage keys are valid. If the problem cannot be resolved, contact your system programmer.

- On z/OS, check the joblog and logrec, and whether any information has been displayed on the console. If this error occurs on an MQCONN or MQCONNX call, check that the subsystem named is an active MQ subsystem. In particular, check that it is not a Db2 subsystem. If the problem cannot be resolved, rerun the application with a CSQSNAP DD card (if you have not already got a dump) and send the resulting dump to IBM.
- On IBM i, consult the FFST record to obtain more detail about the problem.
- On HP Integrity NonStop Server, and UNIX systems, consult the FDC file to obtain more detail about the problem.

2196 (0894) (RC2196): MQRC_UNKNOWN_XMIT_Q:

Explanation

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or the *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (in the latter case queue-manager aliasing is being used), but the *XmitQName* attribute of the definition is not blank and not the name of a locally-defined queue.

Completion Code

MQCC_FAILED

Programmer response

Check the values specified for *ObjectName* and *ObjectQMgrName*. If these are correct, check the queue definitions.

This reason code is also used to identify the corresponding event message Unknown Transmission Queue.

2197 (0895) (RC2197): MQRC_UNKNOWN_DEF_XMIT_Q:

Explanation

An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. If a local definition of the remote queue was specified, or if a queue-manager alias is being resolved, the *XmitQName* attribute in the local definition is blank.

Because there is no queue defined with the same name as the destination queue manager, the queue manager has attempted to use the default transmission queue. However, the name defined by the *DefXmitQName* queue-manager attribute is not the name of a locally-defined queue.

Completion Code

MQCC_FAILED

Programmer response

Correct the queue definitions, or the queue-manager attribute.

This reason code is also used to identify the corresponding event message Unknown Default Transmission Queue.

2198 (0896) (RC2198): MQRC_DEF_XMIT_Q_TYPE_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank.

Because there is no transmission queue defined with the same name as the destination queue manager, the local queue manager has attempted to use the default transmission queue. However, although there is a queue defined by the *DefXmitQName* queue-manager attribute, it is not a local queue.

Completion Code

MQCC_FAILED

Programmer response

Do one of the following:
- Specify a local transmission queue as the value of the *XmitQName* attribute in the local definition of the remote queue.
- Define a local transmission queue with a name that is the same as that of the remote queue manager.
- Specify a local transmission queue as the value of the *DefXmitQName* queue-manager attribute.

See XmitQName for more information about transmission queue names.

This reason code is also used to identify the corresponding event message Default Transmission Queue Type Errorr.

2199 (0897) (RC2199): MQRC_DEF_XMIT_Q_USAGE_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank.

Because there is no transmission queue defined with the same name as the destination queue manager, the local queue manager has attempted to use the default transmission queue. However, the queue defined by the *DefXmitQName* queue-manager attribute does not have a *Usage* attribute of MQUS_TRANSMISSION.

This reason code is returned from MQOPEN or MQPUT1, if the queue manager's Default Transmission Queue is about to be used, but the name of this queue is SYSTEM.CLUSTER.TRANSMIT.QUEUE. This queue is reserved for clustering, so it is not valid to set the queue manager's Default Transmission Queue to this name.

Completion Code

MQCC_FAILED

Programmer response

Do one of the following:
- Specify a local transmission queue as the value of the *XmitQName* attribute in the local definition of the remote queue.
- Define a local transmission queue with a name that is the same as that of the remote queue manager.
- Specify a different local transmission queue as the value of the *DefXmitQName* queue-manager attribute.
- Change the *Usage* attribute of the *DefXmitQName* queue to MQUS_TRANSMISSION.

See XmitQName for more information about transmission queue names.

This reason code is also used to identify the corresponding event message Default Transmission Queue Usage Error.

2201 (0899) (RC2201): MQRC_NAME_IN_USE:

Explanation

An MQOPEN call was issued to create a dynamic queue, but a queue with the same name as the dynamic queue already exists. The existing queue is one that is logically deleted, but for which there are still one or more open handles. For more information, see MQOPEN.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Either ensure that all handles for the previous dynamic queue are closed, or ensure that the name of the new queue is unique; see the description for reason code MQRC_OBJECT_ALREADY_EXISTS.

2202 (089A) (RC2202): MQRC_CONNECTION_QUIESCING:

Explanation

This reason code is issued when the connection to the queue manager is in quiescing state, and an application issues one of the following calls:

- MQCONN or MQCONNX
- MQOPEN, with no connection established, or with MQOO_FAIL_IF_QUIESCING included in the *Options* parameter
- MQGET, with MQGMO_FAIL_IF_QUIESCING included in the *Options* field of the *GetMsgOpts* parameter
- MQPUT or MQPUT1, with MQPMO_FAIL_IF_QUIESCING included in the *Options* field of the *PutMsgOpts* parameter

MQRC_CONNECTION_QUIESCING is also issued by the message channel agent (MCA) when the queue manager is in quiescing state.

Completion Code

MQCC_FAILED

Programmer response

The application should tidy up and terminate. Any uncommitted changes in a unit of work should be backed out.

2203 (089B) (RC2203): MQRC_CONNECTION_STOPPING:

Explanation

This reason code is issued when the connection to the queue manager is shutting down, and the application issues an MQI call. No more message-queuing calls can be issued. For the MQGET call, if the MQGMO_WAIT option was specified, the wait is canceled.

Note that the MQRC_CONNECTION_BROKEN reason may be returned instead if, as a result of system scheduling factors, the queue manager shuts down before the call completes.

MQRC_CONNECTION_STOPPING is also issued by the message channel agent (MCA) when the queue manager is shutting down.

For MQ MQI client applications, it is possible that the call did complete successfully, even though this reason code is returned with a *CompCode* of MQCC_FAILED.

Completion Code

MQCC_FAILED

Programmer response

The application should tidy up and terminate. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

2204 (089C) (RC2204): MQRC_ADAPTER_NOT_AVAILABLE:

Explanation

This is issued only for CICS applications, if any call is issued and the CICS adapter (a Task Related User Exit) has been disabled, or has not been enabled.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

The application should tidy up and terminate. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

2206 (089E) (RC2206): MQRC_MSG_ID_ERROR:

Explanation

An MQGET call was issued to retrieve a message using the message identifier as a selection criterion, but the call failed because selection by message identifier is not supported on this queue.

- On z/OS, the queue is a shared queue, but the *IndexType* queue attribute does not have an appropriate value:
  - If selection is by message identifier alone, *IndexType* must have the value MQIT_MSG_ID.
  - If selection is by message identifier and correlation identifier combined, *IndexType* must have the value MQIT_MSG_ID or MQIT_CORREL_ID. However, the match-any values of MQCI_NONE and MQMI_NONE respectively are exceptions to this rule, and result in the 2206 MQRC_MSG_ID_ERROR reason code.
- On HP Integrity NonStop Server, a key file is required but has not been defined.

Completion Code

MQCC_FAILED

Programmer response

Do one of the following:

- Modify the application so that it does not use selection by message identifier: set the *MsgId* field to MQMI_NONE and do not specify MQMO_MATCH_MSG_ID in MQGMO.
- On z/OS, change the *IndexType* queue attribute to MQIT_MSG_ID.
- On HP Integrity NonStop Server, define a key file.

2207 (089F) (RC2207): MQRC_CORREL_ID_ERROR:

Explanation

An MQGET call was issued to retrieve a message using the correlation identifier as a selection criterion, but the call failed because selection by correlation identifier is not supported on this queue.

- On z/OS, the queue is a shared queue, but the *IndexType* queue attribute does not have an appropriate value:
  - If selection is by correlation identifier alone, *IndexType* must have the value MQIT_CORREL_ID.
  - If selection is by correlation identifier and message identifier combined, *IndexType* must have the value MQIT_CORREL_ID or MQIT_MSG_ID.
- On HP Integrity NonStop Server, a key file is required but has not been defined.

Completion Code

MQCC_FAILED

Programmer response

Do one of the following:

- On z/OS, change the *IndexType* queue attribute to MQIT_CORREL_ID.
- On HP Integrity NonStop Server, define a key file.
- Modify the application so that it does not use selection by correlation identifier: set the *CorrelId* field to MQCI_NONE and do not specify MQMO_MATCH_CORREL_ID in MQGMO.

2208 (08A0) (RC2208): MQRC_FILE_SYSTEM_ERROR:

Explanation

An unexpected return code was received from the file system, in attempting to perform an operation on a queue.

This reason code occurs only on VSE/ESA.

Completion Code

MQCC_FAILED

Programmer response

Check the file system definition for the queue that was being accessed. For a VSAM file, check that the control interval is large enough for the maximum message length allowed for the queue.

2209 (08A1) (RC2209): MQRC_NO_MSG_LOCKED:

Explanation

An MQGET call was issued with the MQGMO_UNLOCK option, but no message was currently locked.

Completion Code

MQCC_WARNING

Programmer response

Check that a message was locked by an earlier MQGET call with the MQGMO_LOCK option for the same handle, and that no intervening call has caused the message to become unlocked.

2210 (08A2) (RC2210): MQRC_SOAP_DOTNET_ERROR:

Explanation

This exception has been received from an external .NET environment. For more information, see the inner exception that is contained within the received exception message.

Completion Code

MQCC_FAILED

Programmer response

Refer to the .NET documentation for information about the inner exception. Follow the corrective action recommended there.

2211 (08A3) (RC2211): MQRC_SOAP_AXIS_ERROR:

Explanation

An exception from the Axis environment has been received and is included as a chained exception.

Completion Code

MQCC_FAILED

Programmer response

Refer to the Axis documentation for details about the chained exception. Follow the corrective action recommended there.

2212 (08A4) (RC2212): MQRC_SOAP_URL_ERROR:

Explanation

The SOAP URL has been specified incorrectly.

Completion Code

MQCC_FAILED

Programmer response

Correct the SOAP URL and rerun.

2217 (08A9) (RC2217): MQRC_CONNECTION_NOT_AUTHORIZED:

Explanation

This reason code occurs only on z/OS.

If the queue manager has been configured to use Advanced Message Security this reason code is returned if an error occurs in security processing.

This reason code might indicate a privacy security policy has been defined for the target queue that does not identify any recipients.

This reason code is also returned to CICS applications if the CICS subsystem is not authorized to connect to the queue manager.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the subsystem is authorized to connect to the queue manager.

2218 (08AA) (RC2218): MQRC_MSG_TOO_BIG_FOR_CHANNEL:

Explanation

A message was put to a remote queue, but the message is larger than the maximum message length allowed by the channel. This reason code is returned in the *Feedback* field in the message descriptor of a report message.

- On z/OS, this return code is issued only if you are not using CICS for distributed queuing. Otherwise, MQRC_MSG_TOO_BIG_FOR_Q_MGR is issued.

Completion Code

MQCC_FAILED

Programmer response

Check the channel definitions. Increase the maximum message length that the channel can accept, or break the message into several smaller messages.

2219 (08AB) (RC2219): MQRC_CALL_IN_PROGRESS:

Explanation

The application issued an MQI call whilst another MQI call was already being processed for that connection. Only one call per application connection can be processed at a time.

Concurrent calls can arise when an application uses multiple threads, or when an exit is invoked as part of the processing of an MQI call. For example, a data-conversion exit invoked as part of the processing of the MQGET call may try to issue an MQI call.

- On z/OS, concurrent calls can arise only with batch or IMS applications; an example is when a subtask ends while an MQI call is in progress (for example, an MQGET that is waiting), and there is an end-of-task exit routine that issues another MQI call.
- On Windows, concurrent calls can also arise if an MQI call is issued in response to a user message while another MQI call is in progress.
- If the application is using multiple threads with shared handles, MQRC_CALL_IN_PROGRESS occurs when the handle specified on the call is already in use by another thread and MQCNO_HANDLE_SHARE_NO_BLOCK was specified on the MQCONNX call.

Completion Code

MQCC_FAILED

Programmer response

Ensure that an MQI call cannot be issued while another one is active. Do not issue MQI calls from within a data-conversion exit.

- On z/OS, if you want to provide a subtask to allow an application that is waiting for a message to arrive to be canceled, wait for the message by using MQGET with MQGMO_SET_SIGNAL, rather than MQGMO_WAIT.

2220 (08AC) (RC2220): MQRC_RMH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQRMH structure that is not valid. Possible errors include the following:

- The *StrucId* field is not MQRMH_STRUC_ID.
- The *Version* field is not MQRMH_VERSION_1.
- The *StrucLength* field specifies a value that is too small to include the structure plus the variable-length data at the end of the structure.
- The *CodedCharSetId* field is zero, or a negative value that is not valid.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the *CodedCharSetId* field to a valid value (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in this field).

2222 (08AE) (RC2222): MQRC_Q_MGR_ACTIVE:

Explanation

This condition is detected when a queue manager becomes active.
- On z/OS, this event is not generated for the first start of a queue manager, only on subsequent restarts.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Queue Manager Active.

2223 (08AF) (RC2223): MQRC_Q_MGR_NOT_ACTIVE:

Explanation

This condition is detected when a queue manager is requested to stop or quiesce.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Queue Manager Not Active.

2224 (08B0) (RC2224): MQRC_Q_DEPTH_HIGH:

Explanation

An MQPUT or MQPUT1 call has caused the queue depth to be incremented to, or greater than, the limit specified in the *QDepthHighLimit* attribute.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Queue Depth High.

2225 (08B1) (RC2225): MQRC_Q_DEPTH_LOW:

Explanation

An MQGET call has caused the queue depth to be decremented to, or less than, the limit specified in the *QDepthLowLimit* attribute.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Queue Depth Low.

2226 (08B2) (RC2226): MQRC_Q_SERVICE_INTERVAL_HIGH:

Explanation

No successful gets or puts have been detected within an interval that is greater than the limit specified in the *QServiceInterval* attribute.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Queue Service Interval High.

2227 (08B3) (RC2227): MQRC_Q_SERVICE_INTERVAL_OK:

Explanation

A successful get has been detected within an interval that is less than or equal to the limit specified in the *QServiceInterval* attribute.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Queue Service Interval OK.

2228 (08B4) (RC2228): MQRC_RFH_HEADER_FIELD_ERROR:

Explanation

An expected RFH header field was not found or had an invalid value. If this error occurs in an IBM MQ SOAP listener, the missing or erroneous field is either the *contentType* field or the *transportVersion* field or both.

Completion Code

MQCC_FAILED

Programmer response

If this error occurs in an IBM MQ SOAP listener, and you are using the IBM-supplied sender, contact your IBM Support Center. If you are using a bespoke sender, check the associated error message, and that the RFH2 section of the SOAP/MQ request message contains all the mandatory fields, and that these fields have valid values.

2229 (08B5) (RC2229): MQRC_RAS_PROPERTY_ERROR:

Explanation

There is an error related to the RAS property file. The file might be missing, it might be not accessible, or the commands in the file might be incorrect.

Completion Code

MQCC_FAILED

Programmer response

Look at the associated error message, which explains the error in detail. Correct the error and try again.

2232 (08B8) (RC2232): MQRC_UNIT_OF_WORK_NOT_STARTED:

Explanation

An MQGET, MQPUT or MQPUT1 call was issued to get or put a message within a unit of work, but no TM/MP transaction had been started. If MQGMO_NO_SYNCPOINT is not specified on MQGET, or MQPMO_NO_SYNCPOINT is not specified on MQPUT or MQPUT1 (the default), the call requires a unit of work.

Completion Code

MQCC_FAILED

Programmer response

Ensure a TM/MP transaction is available, or issue the MQGET call with the MQGMO_NO_SYNCPOINT option, or the MQPUT or MQPUT1 call with the MQPMO_NO_SYNCPOINT option, which will cause a transaction to be started automatically.

2233 (08B9) (RC2233): MQRC_CHANNEL_AUTO_DEF_OK:

Explanation

This condition is detected when the automatic definition of a channel is successful. The channel is defined by the MCA.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Auto-definition OK.

2234 (08BA) (RC2234): MQRC_CHANNEL_AUTO_DEF_ERROR:

Explanation

This condition is detected when the automatic definition of a channel fails; this might be because an error occurred during the definition process, or because the channel automatic-definition exit inhibited the definition. Additional information is returned in the event message indicating the reason for the failure.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING

Programmer response

This reason code is only used to identify the corresponding event message Channel Auto-definition Error.

Examine the additional information returned in the event message to determine the reason for the failure.

2235 (08BB) (RC2235): MQRC_CFH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFH structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2236 (08BC) (RC2236): MQRC_CFIL_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFIL or MQRCFIL64 structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2237 (08BD) (RC2237): MQRC_CFIN_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFIN or MQCFIN64 structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2238 (08BE) (RC2238): MQRC_CFSL_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFSL structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2239 (08BF) (RC2239): MQRC_CFST_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFST structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2241 (08C1) (RC2241): MQRC_INCOMPLETE_GROUP:

Explanation

An operation was attempted on a queue using a queue handle that had an incomplete message group. This reason code can arise in the following situations:

- On the MQPUT call, when the application specifies MQPMO_LOGICAL_ORDER and attempts to put a message that is not in a group. The completion code is MQCC_FAILED in this case.
- On the MQPUT call, when the application does *not* specify MQPMO_LOGICAL_ORDER, but the previous MQPUT call for the queue handle did specify MQPMO_LOGICAL_ORDER. The completion code is MQCC_WARNING in this case.
- On the MQGET call, when the application does *not* specify MQGMO_LOGICAL_ORDER, but the previous MQGET call for the queue handle did specify MQGMO_LOGICAL_ORDER. The completion code is MQCC_WARNING in this case.
- On the MQCLOSE call, when the application attempts to close the queue that has the incomplete message group. The completion code is MQCC_WARNING in this case.

If there is an incomplete logical message as well as an incomplete message group, reason code MQRC_INCOMPLETE_MSG is returned in preference to MQRC_INCOMPLETE_GROUP.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

If this reason code is expected, no corrective action is required. Otherwise, ensure that the MQPUT call for the last message in the group specifies MQMF_LAST_MSG_IN_GROUP.

2242 (08C2) (RC2242): MQRC_INCOMPLETE_MSG:

Explanation

An operation was attempted on a queue using a queue handle that had an incomplete logical message. This reason code can arise in the following situations:

- On the MQPUT call, when the application specifies MQPMO_LOGICAL_ORDER and attempts to put a message that is not a segment, or that has a setting for the MQMF_LAST_MSG_IN_GROUP flag that is different from the previous message. The completion code is MQCC_FAILED in this case.
- On the MQPUT call, when the application does *not* specify MQPMO_LOGICAL_ORDER, but the previous MQPUT call for the queue handle did specify MQPMO_LOGICAL_ORDER. The completion code is MQCC_WARNING in this case.
- On the MQGET call, when the application does *not* specify MQGMO_LOGICAL_ORDER, but the previous MQGET call for the queue handle did specify MQGMO_LOGICAL_ORDER. The completion code is MQCC_WARNING in this case.
- On the MQCLOSE call, when the application attempts to close the queue that has the incomplete logical message. The completion code is MQCC_WARNING in this case.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

If this reason code is expected, no corrective action is required. Otherwise, ensure that the MQPUT call for the last segment specifies MQMF_LAST_SEGMENT.

2243 (08C3) (RC2243): MQRC_INCONSISTENT_CCSIDS:

Explanation

An MQGET call was issued specifying the MQGMO_COMPLETE_MSG option, but the message to be retrieved consists of two or more segments that have differing values for the `CodedCharSetId` field in MQMD. This can arise when the segments take different paths through the network, and some of those paths have MCA sender conversion enabled. The call succeeds with a completion code of MQCC_WARNING, but only the first few segments that have identical character-set identifiers are returned.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING

Programmer response

Remove the MQGMO_COMPLETE_MSG option from the MQGET call and retrieve the remaining message segments one by one.

2244 (08C4) (RC2244): MQRC_INCONSISTENT_ENCODINGS:

Explanation

An MQGET call was issued specifying the MQGMO_COMPLETE_MSG option, but the message to be retrieved consists of two or more segments that have differing values for the *Encoding* field in MQMD. This can arise when the segments take different paths through the network, and some of those paths have MCA sender conversion enabled. The call succeeds with a completion code of MQCC_WARNING, but only the first few segments that have identical encodings are returned.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING

Programmer response

Remove the MQGMO_COMPLETE_MSG option from the MQGET call and retrieve the remaining message segments one by one.

2245 (08C5) (RC2245): MQRC_INCONSISTENT_UOW:

Explanation

One of the following applies:

- An MQPUT call was issued to put a message in a group or a segment of a logical message, but the value specified or defaulted for the MQPMO_SYNCPOINT option is not consistent with the current group and segment information retained by the queue manager for the queue handle.

  If the current call specifies MQPMO_LOGICAL_ORDER, the call fails. If the current call does not specify MQPMO_LOGICAL_ORDER, but the previous MQPUT call for the queue handle did, the call succeeds with completion code MQCC_WARNING.
- An MQGET call was issued to remove from the queue a message in a group or a segment of a logical message, but the value specified or defaulted for the MQGMO_SYNCPOINT option is not consistent with the current group and segment information retained by the queue manager for the queue handle.

  If the current call specifies MQGMO_LOGICAL_ORDER, the call fails. If the current call does not specify MQGMO_LOGICAL_ORDER, but the previous MQGET call for the queue handle did, the call succeeds with completion code MQCC_WARNING.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

Modify the application to ensure that the same unit-of-work specification is used for all messages in the group, or all segments of the logical message.

2246 (08C6) (RC2246): MQRC_INVALID_MSG_UNDER_CURSOR:

Explanation

An MQGET call was issued specifying the MQGMO_COMPLETE_MSG option with either MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR, but the message that is under the cursor has an MQMD with an `Offset` field that is greater than zero. Because MQGMO_COMPLETE_MSG was specified, the message is not valid for retrieval.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Reposition the browse cursor so that it is located on a message with an `Offset` field in MQMD that is zero. Alternatively, remove the MQGMO_COMPLETE_MSG option.

2247 (08C7) (RC2247): MQRC_MATCH_OPTIONS_ERROR:

Explanation

An MQGET call was issued, but the value of the `MatchOptions` field in the `GetMsgOpts` parameter is not valid, for one of the following reasons:
- An undefined option is specified.
- All of the following are true:
  - MQGMO_LOGICAL_ORDER is specified.
  - There is a current message group or logical message for the queue handle.
  - Neither MQGMO_BROWSE_MSG_UNDER_CURSOR nor MQGMO_MSG_UNDER_CURSOR is specified.
  - One or more of the MQMO_* options is specified.
  - The values of the fields in the `MsgDesc` parameter corresponding to the MQMO_* options specified, differ from the values of those fields in the MQMD for the message to be returned next.
- On z/OS, one or more of the options specified is not valid for the index type of the queue.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that only valid options are specified for the field.

2248 (08C8) (RC2248): MQRC_MDE_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQMDE structure that is not valid. Possible errors include the following:

- The `StrucId` field is not MQMDE_STRUC_ID.
- The `Version` field is not MQMDE_VERSION_2.
- The `StrucLength` field is not MQMDE_LENGTH_2.
- The `CodedCharSetId` field is zero, or a negative value that is not valid.
- The `BufferLength` parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the `CodedCharSetId` field to a valid value (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in this field).

2249 (08C9) (RC2249): MQRC_MSG_FLAGS_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the `MsgFlags` field in the message descriptor MQMD contains one or more message flags that are not recognized by the local queue manager. The message flags that cause this reason code to be returned depend on the destination of the message; see the description of REPORT in Report options and message flags for more information.

This reason code can also occur in the `Feedback` field in the MQMD of a report message, or in the `Reason` field in the MQDLH structure of a message on the dead-letter queue; in both cases it indicates that the destination queue manager does not support one or more of the message flags specified by the sender of the message.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Do the following:

- Ensure that the `MsgFlags` field in the message descriptor is initialized with a value when the message descriptor is declared, or is assigned a value prior to the MQPUT or MQPUT1 call. Specify MQMF_NONE if no message flags are needed.

- Ensure that the message flags specified are valid; see the *MsgFlags* field described in the description of MQMD in MsgFlags (MQLONG) for valid message flags.
- If multiple message flags are being set by adding the individual message flags together, ensure that the same message flag is not added twice.
- On z/OS, ensure that the message flags specified are valid for the index type of the queue; see the description of the *MsgFlags* field in MQMD for further details.

2250 (08CA) (RC2250): MQRC_MSG_SEQ_NUMBER_ERROR:

Explanation

An MQGET, MQPUT, or MQPUT1 call was issued, but the value of the *MsgSeqNumber* field in the MQMD or MQMDE structure is less than one or greater than 999 999 999.

This error can also occur on the MQPUT call if the *MsgSeqNumber* field would have become greater than 999 999 999 as a result of the call.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify a value in the range 1 through 999 999 999. Do not attempt to create a message group containing more than 999 999 999 messages.

2251 (08CB) (RC2251): MQRC_OFFSET_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the value of the *Offset* field in the MQMD or MQMDE structure is less than zero or greater than 999 999 999.

This error can also occur on the MQPUT call if the *Offset* field would have become greater than 999 999 999 as a result of the call.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify a value in the range 0 through 999 999 999. Do not attempt to create a message segment that would extend beyond an offset of 999 999 999.

2252 (08CC) (RC2252): MQRC_ORIGINAL_LENGTH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued to put a report message that is a segment, but the `OriginalLength` field in the MQMD or MQMDE structure is either:

- Less than the length of data in the message, or
- Less than one (for a segment that is not the last segment), or
- Less than zero (for a segment that is the last segment)

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify a value that is greater than zero. Zero is valid only for the last segment.

2253 (08CD) (RC2253): MQRC_SEGMENT_LENGTH_ZERO:

Explanation

An MQPUT or MQPUT1 call was issued to put the first or an intermediate segment of a logical message, but the length of the application message data in the segment (excluding any MQ headers that may be present) is zero. The length must be at least one for the first or intermediate segment.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check the application logic to ensure that segments are put with a length of one or greater. Only the last segment of a logical message is permitted to have a length of zero.

2255 (08CF) (RC2255): MQRC_UOW_NOT_AVAILABLE:

Explanation

An MQGET, MQPUT, or MQPUT1 call was issued to get or put a message outside a unit of work, but the options specified on the call required the queue manager to process the call within a unit of work. Because there is already a user-defined unit of work in existence, the queue manager was unable to create a temporary unit of work for the duration of the call.

This reason occurs in the following circumstances:
- On an MQGET call, when the MQGMO_COMPLETE_MSG option is specified in MQGMO and the logical message to be retrieved is persistent and consists of two or more segments.
- On an MQPUT or MQPUT1 call, when the MQMF_SEGMENTATION_ALLOWED flag is specified in MQMD and the message requires segmentation.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Issue the MQGET, MQPUT, or MQPUT1 call inside the user-defined unit of work. Alternatively, for the MQPUT or MQPUT1 call, reduce the size of the message so that it does not require segmentation by the queue manager.

2256 (08D0) (RC2256): MQRC_WRONG_GMO_VERSION:

Explanation

An MQGET call was issued specifying options that required an MQGMO with a version number not less than MQGMO_VERSION_2, but the MQGMO supplied did not satisfy this condition.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Modify the application to pass a version-2 MQGMO. Check the application logic to ensure that the *Version* field in MQGMO has been set to MQGMO_VERSION_2. Alternatively, remove the option that requires the version-2 MQGMO.

2257 (08D1) (RC2257): MQRC_WRONG_MD_VERSION:

Explanation

An MQGET, MQPUT, or MQPUT1 call was issued specifying options that required an MQMD with a version number not less than MQMD_VERSION_2, but the MQMD supplied did not satisfy this condition.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Modify the application to pass a version-2 MQMD. Check the application logic to ensure that the `Version` field in MQMD has been set to MQMD_VERSION_2. Alternatively, remove the option that requires the version-2 MQMD.

2258 (08D2) (RC2258): MQRC_GROUP_ID_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued to put a distribution-list message that is also a message in a group, a message segment, or has segmentation allowed, but an invalid combination of options and values was specified. All of the following are true:

- MQPMO_LOGICAL_ORDER is not specified in the `Options` field in MQPMO.
- Either there are too few MQPMR records provided by MQPMO, or the `GroupId` field is not present in the MQPMR records.
- One or more of the following flags is specified in the `MsgFlags` field in MQMD or MQMDE:
  - MQMF_SEGMENTATION_ALLOWED
  - MQMF_*_MSG_IN_GROUP
  - MQMF_*_SEGMENT
- The `GroupId` field in MQMD or MQMDE is not MQGI_NONE.

This combination of options and values would result in the same group identifier being used for all of the destinations in the distribution list; this is not permitted by the queue manager.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify MQGI_NONE for the `GroupId` field in MQMD or MQMDE. Alternatively, if the call is MQPUT specify MQPMO_LOGICAL_ORDER in the `Options` field in MQPMO.

2259 (08D3) (RC2259): MQRC_INCONSISTENT_BROWSE:

Explanation

An MQGET call was issued with the MQGMO_BROWSE_NEXT option specified, but the specification of the MQGMO_LOGICAL_ORDER option for the call is different from the specification of that option for the previous call for the queue handle. Either both calls must specify MQGMO_LOGICAL_ORDER, or neither call must specify MQGMO_LOGICAL_ORDER.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Add or remove the MQGMO_LOGICAL_ORDER option as appropriate. Alternatively, to switch between logical order and physical order, specify the MQGMO_BROWSE_FIRST option to restart the scan from the beginning of the queue, omitting or specifying MQGMO_LOGICAL_ORDER as required.

2260 (08D4) (RC2260): MQRC_XQH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQXQH structure that is not valid. Possible errors include the following:
- The *StrucId* field is not MQXQH_STRUC_ID.
- The *Version* field is not MQXQH_VERSION_1.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2261 (08D5) (RC2261): MQRC_SRC_ENV_ERROR:

Explanation

This reason occurs when a channel exit that processes reference messages detects an error in the source environment data of a reference message header (MQRMH). One of the following is true:

- *SrcEnvLength* is less than zero.
- *SrcEnvLength* is greater than zero, but there is no source environment data.
- *SrcEnvLength* is greater than zero, but *SrcEnvOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *SrcEnvLength* is greater than zero, but *SrcEnvOffset* plus *SrcEnvLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify the source environment data correctly.

2262 (08D6) (RC2262): MQRC_SRC_NAME_ERROR:

Explanation

This reason occurs when a channel exit that processes reference messages detects an error in the source name data of a reference message header (MQRMH). One of the following is true:

- *SrcNameLength* is less than zero.
- *SrcNameLength* is greater than zero, but there is no source name data.
- *SrcNameLength* is greater than zero, but *SrcNameOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *SrcNameLength* is greater than zero, but *SrcNameOffset* plus *SrcNameLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify the source name data correctly.

2263 (08D7) (RC2263): MQRC_DEST_ENV_ERROR:

Explanation

This reason occurs when a channel exit that processes reference messages detects an error in the destination environment data of a reference message header (MQRMH). One of the following is true:

- *DestEnvLength* is less than zero.
- *DestEnvLength* is greater than zero, but there is no destination environment data.
- *DestEnvLength* is greater than zero, but *DestEnvOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *DestEnvLength* is greater than zero, but *DestEnvOffset* plus *DestEnvLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify the destination environment data correctly.

2264 (08D8) (RC2264): MQRC_DEST_NAME_ERROR:

Explanation

This reason occurs when a channel exit that processes reference messages detects an error in the destination name data of a reference message header (MQRMH). One of the following is true:

- *DestNameLength* is less than zero.
- *DestNameLength* is greater than zero, but there is no destination name data.
- *DestNameLength* is greater than zero, but *DestNameOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *DestNameLength* is greater than zero, but *DestNameOffset* plus *DestNameLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Specify the destination name data correctly.

2265 (08D9) (RC2265): MQRC_TM_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQTM structure that is not valid. Possible errors include the following:

- The *StrucId* field is not MQTM_STRUC_ID.
- The *Version* field is not MQTM_VERSION_1.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2266 (08DA) (RC2266): MQRC_CLUSTER_EXIT_ERROR:

Explanation

An MQOPEN, MQPUT, or MQPUT1 call was issued to open or put a message on a cluster queue, but the cluster workload exit defined by the queue-manager's *ClusterWorkloadExit* attribute failed unexpectedly or did not respond in time. Subsequent MQOPEN, MQPUT, and MQPUT1 calls for this queue handle are processed as though the *ClusterWorkloadExit* attribute were blank.

- On z/OS, a message giving more information about the error is written to the system log, for example message CSQV455E or CSQV456E.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check the cluster workload exit to ensure that it has been written correctly.

2267 (08DB) (RC2267): MQRC_CLUSTER_EXIT_LOAD_ERROR:

Explanation

An MQCONN or MQCONNX call was issued to connect to a queue manager, but the queue manager was unable to load the cluster workload exit. Execution continues without the cluster workload exit.
- On z/OS, if the cluster workload exit cannot be loaded, a message is written to the system log, for example message CSQV453I. Processing continues as though the *ClusterWorkloadExit* attribute had been blank.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_WARNING

Programmer response

Ensure that the queue-manager's *ClusterWorkloadExit* attribute has the correct value, and that the exit has been installed into the correct location.

2268 (08DC) (RC2268): MQRC_CLUSTER_PUT_INHIBITED:

Explanation

An MQOPEN call with the MQOO_OUTPUT and MQOO_BIND_ON_OPEN options in effect was issued for a cluster queue, but the call failed because all of the following are true:
- All instances of the cluster queue are currently put-inhibited (that is, all of the queue instances have the *InhibitPut* attribute set to MQQA_PUT_INHIBITED).
- There is no local instance of the queue. (If there is a local instance, the MQOPEN call succeeds, even if the local instance is put-inhibited.)
- There is no cluster workload exit for the queue, or there is a cluster workload exit but it did not choose a queue instance. (If the cluster workload exit does choose a queue instance, the MQOPEN call succeeds, even if that instance is put-inhibited.)

If the MQOO_BIND_NOT_FIXED option is specified on the MQOPEN call, the call can succeed even if all of the queues in the cluster are put-inhibited. However, a subsequent MQPUT call may fail if all of the queues are still put-inhibited at the time of the MQPUT call.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

If the system design allows put requests to be inhibited for short periods, retry the operation later. If the problem persists, determine why all of the queues in the cluster are put-inhibited.

2269 (08DD) (RC2269): MQRC_CLUSTER_RESOURCE_ERROR:

Explanation

An MQOPEN, MQPUT, or MQPUT1 call was issued for a cluster queue, but an error occurred whilst trying to use a resource required for clustering.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Do the following:
- Check that the SYSTEM.CLUSTER.* queues are not put inhibited or full.
- Check the event queues for any events relating to the SYSTEM.CLUSTER.* queues, as these may give guidance as to the nature of the failure.
- Check that the repository queue manager is available.
- On z/OS, check the console for signs of the failure, such as full page sets.

2270 (08DE) (RC2270): MQRC_NO_DESTINATIONS_AVAILABLE:

Explanation

An MQPUT or MQPUT1 call was issued to put a message on a cluster queue, but at the time of the call there were no longer any instances of the queue in the cluster. The message therefore could not be sent.

This situation can occur when MQOO_BIND_NOT_FIXED is specified on the MQOPEN call that opens the queue, or MQPUT1 is used to put the message.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check the queue definition and queue status to determine why all instances of the queue were removed from the cluster. Correct the problem and rerun the application.

2271 (08DF) (RC2271): MQRC_CONN_TAG_IN_USE:

Explanation

An MQCONNX call was issued specifying one of the MQCNO_*_CONN_TAG_* options, but the call failed because the connection tag specified by *ConnTag* in MQCNO is in use by an active process or thread, or there is an unresolved unit of work that references this connection tag.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

The problem is likely to be transitory. The application should wait a short while and then retry the operation.

2272 (08E0) (RC2272): MQRC_PARTIALLY_CONVERTED:

Explanation

On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, one or more MQ header structures in the message data could not be converted to the specified target character set or encoding. In this situation, the MQ header structures are converted to the queue-manager's character set and encoding, and the application data in the message is converted to the target character set and encoding. On return from the call, the values returned in the various *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter and MQ header structures indicate the character set and encoding that apply to each part of the message. The call completes with MQCC_WARNING.

This reason code usually occurs when the specified target character set is one that causes the character strings in the MQ header structures to expand beyond the lengths of their fields. Unicode character set UCS-2 is an example of a character set that causes this to happen.

Completion Code

MQCC_FAILED

Programmer response

If this is an expected situation, no corrective action is required.

If this is an unexpected situation, check that the MQ header structures contain valid data. If they do, specify as the target character set a character set that does not cause the strings to expand.

2273 (08E1) (RC2273): MQRC_CONNECTION_ERROR:

Explanation

An MQCONN or MQCONNX call failed for one of the following reasons:

- The installation and customization options chosen for IBM MQ do not allow connection by the type of application being used.
- The system parameter module is not at the same release level as the queue manager.
- The channel initiator is not at the same release level as the queue manager.
- An internal error was detected by the queue manager.

Completion Code

MQCC_FAILED

Programmer response

None, if the installation and customization options chosen for IBM MQ do not allow all functions to be used.

Otherwise, if this occurs while starting the channel initiator, ensure that the queue manager and the channel initiator are both at the same release level and that their started task JCL procedures both specify the same level of IBM MQ program libraries; if this occurs while starting the queue manager, relinkedit the system parameter module (CSQZPARM) to ensure that it is at the correct level. If the problem persists, contact your IBM support center.

2274 (08E2) (RC2274): MQRC_OPTION_ENVIRONMENT_ERROR:

Explanation

An MQGET call with the MQGMO_MARK_SKIP_BACKOUT option specified was issued from a Db2 Stored Procedure. The call failed because the MQGMO_MARK_SKIP_BACKOUT option cannot be used from a Db2 Stored Procedure.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Remove the MQGMO_MARK_SKIP_BACKOUT option from the MQGET call.

2277 (08E5) (RC2277): MQRC_CD_ERROR:

Explanation

An MQCONNX call was issued to connect to a queue manager, but the MQCD channel definition structure addressed by the `ClientConnOffset` or `ClientConnPtr` field in MQCNO contains data that is not valid. Consult the error log for more information about the nature of the error.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Ensure that input fields in the MQCD structure are set correctly.

2278 (08E6) (RC2278): MQRC_CLIENT_CONN_ERROR:

Explanation

An MQCONNX call was issued to connect to a queue manager, but the MQCD channel definition structure is not specified correctly. One of the following applies:

- `ClientConnOffset` is not zero and `ClientConnPtr` is not zero and not the null pointer.
- `ClientConnPtr` is not a valid pointer.
- `ClientConnPtr` or `ClientConnOffset` points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems. It also occurs in Java applications when a client channel definition table (CCDT) is specified to determine the name of the channel, but the table itself cannot be found.

Completion Code

MQCC_FAILED

Programmer response

Ensure that at least one of `ClientConnOffset` and `ClientConnPtr` is zero. Ensure that the field used points to accessible storage. Ensure that the URL of the client channel definition table is correct.

2279 (08E7) (RC2279): MQRC_CHANNEL_STOPPED_BY_USER:

Explanation

This condition is detected when the channel has been stopped by an operator. The reason qualifier identifies the reasons for stopping.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Stopped By User.

2280 (08E8) (RC2280): MQRC_HCONFIG_ERROR:

Explanation

The configuration handle `Hconfig` specified on the MQXEP call or MQZEP call is not valid. The MQXEP call is issued by an API exit function; the MQZEP call is issued by an installable service.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Specify the configuration handle that was provided by the queue manager:
- On the MQXEP call, use the handle passed in the `Hconfig` field of the MQAXP structure.
- On the MQZEP call, use the handle passed to the installable service's configuration function on the component initialization call. For more information about installable services, see Installable services and components for UNIX, Linux and Windows .

2281 (08E9) (RC2281): MQRC_FUNCTION_ERROR:

Explanation

An MQXEP or MQZEP call was issued, but the function identifier `Function` specified on the call is not valid, or not supported by the installable service being configured.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Do the following:
- For the MQXEP call, specify one of the MQXF_* values.
- For the MQZEP call, specify an MQZID_* value that is valid for the installable service being configured. See MQZEP to determine which values are valid.

2282 (08EA) (RC2282): MQRC_CHANNEL_STARTED:

Explanation

One of the following has occurred:
- An operator has issued a Start Channel command.
- An instance of a channel has been successfully established. This condition is detected when Initial Data negotiation is complete and resynchronization has been performed where necessary such that message transfer can proceed.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Started.

2283 (08EB) (RC2283): MQRC_CHANNEL_STOPPED:

Explanation

This condition is detected when the channel has been stopped. The reason qualifier identifies the reasons for stopping.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Stopped.

2284 (08EC) (RC2284): MQRC_CHANNEL_CONV_ERROR:

Explanation

This condition is detected when a channel is unable to do data conversion and the MQGET call to get a message from the transmission queue resulted in a data conversion error. The conversion reason code identifies the reason for the failure.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Conversion Error.

2285 (08ED) (RC2285): MQRC_SERVICE_NOT_AVAILABLE:

Explanation

This reason should be returned by an installable service component when the requested action cannot be performed because the required underlying service is not available.

- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Make the underlying service available.

2286 (08EE) (RC2286): MQRC_INITIALIZATION_FAILED:

Explanation

This reason should be returned by an installable service component when the component is unable to complete initialization successfully.

- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Correct the error and retry the operation.

2287 (08EF) (RC2287): MQRC_TERMINATION_FAILED:

Explanation

This reason should be returned by an installable service component when the component is unable to complete termination successfully.

- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Correct the error and retry the operation.

2288 (08F0) (RC2288): MQRC_UNKNOWN_Q_NAME:

Explanation

This reason should be returned by the MQZ_LOOKUP_NAME installable service component when the name specified for the *QName* parameter is not recognized.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

None. See Installable services and components for UNIX, Linux and Windows for more information about installable services.

2289 (08F1) (RC2289): MQRC_SERVICE_ERROR:

Explanation

This reason should be returned by an installable service component when the component encounters an unexpected error.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Correct the error and retry the operation.

2290 (08F2) (RC2290): MQRC_Q_ALREADY_EXISTS:

Explanation

This reason should be returned by the MQZ_INSERT_NAME installable service component when the queue specified by the *QName* parameter is already defined to the name service.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

None. See Installable services and components for UNIX, Linux and Windows for more information about installable services.

2291 (08F3) (RC2291): MQRC_USER_ID_NOT_AVAILABLE:

Explanation

This reason should be returned by the MQZ_FIND_USERID installable service component when the user ID cannot be determined.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

None. See Installable services and components for UNIX, Linux and Windows for more information about installable services.

2292 (08F4) (RC2292): MQRC_UNKNOWN_ENTITY:

Explanation

This reason should be returned by the authority installable service component when the name specified by the *EntityName* parameter is not recognized.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the entity is defined.

2294 (08F6) (RC2294): MQRC_UNKNOWN_REF_OBJECT:

Explanation

This reason should be returned by the MQZ_COPY_ALL_AUTHORITY installable service component when the name specified by the *RefObjectName* parameter is not recognized.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the reference object is defined. See Installable services and components for UNIX, Linux and Windows for more information about installable services.

2295 (08F7) (RC2295): MQRC_CHANNEL_ACTIVATED:

Explanation

This condition is detected when a channel that has been waiting to become active, and for which a Channel Not Activated event has been generated, is now able to become active because an active slot has been released by another channel.

This event is not generated for a channel that is able to become active without waiting for an active slot to be released.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Activated.

2296 (08F8) (RC2296): MQRC_CHANNEL_NOT_ACTIVATED:

Explanation

This condition is detected when a channel is required to become active, either because it is starting or because it is about to make another attempt to establish connection with its partner. However, it is unable to do so because the limit on the number of active channels has been reached.

- On z/OS, the maximum number of active channels is given by the ACTCHL queue manager attribute.
- In other environments, the maximum number of active channels is given by the MaxActiveChannels parameter in the qm.ini file.

The channel waits until it is able to take over an active slot released when another channel ceases to be active. At that time a Channel Activated event is generated.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Not Activated.

2297 (08F9) (RC2297): MQRC_UOW_CANCELED:

Explanation

An MQI call was issued, but the unit of work (TM/MP transaction) being used for the MQ operation had been canceled. This may have been done by TM/MP itself (for example, due to the transaction running for too long, or exceeding audit trail sizes), or by the application program issuing an ABORT_TRANSACTION. All updates performed to resources owned by the queue manager are backed out.

Completion Code

MQCC_FAILED

Programmer response

Refer to the operating system's *Transaction Management Operations Guide* to determine how the Transaction Manager can be tuned to avoid the problem of system limits being exceeded.

2298 (08FA) (RC2298): MQRC_FUNCTION_NOT_SUPPORTED:

Explanation

The function requested is not available in the current environment.

Completion Code

MQCC_FAILED

Programmer response

Remove the call from the application.

This reason code can be used when the call requires resources or functionality that is restricted by the queue manager OPMODE setting.

If you get this reason code with CICS group connect, check that the queue manager attribute GROUPUR is enabled.

2299 (08FB) (RC2299): MQRC_SELECTOR_TYPE_ERROR:

Explanation

The *`Selector`* parameter has the wrong data type; it must be of type `Long`.

Completion Code

MQCC_FAILED

Programmer response

Declare the *`Selector`* parameter as `Long`.

2300 (08FC) (RC2300): MQRC_COMMAND_TYPE_ERROR:

Explanation

The mqExecute call was issued, but the value of the MQIASY_TYPE data item in the administration bag is not MQCFT_COMMAND.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the MQIASY_TYPE data item in the administration bag has the value MQCFT_COMMAND.

2301 (08FD) (RC2301): MQRC_MULTIPLE_INSTANCE_ERROR:

Explanation

The *Selector* parameter specifies a system selector (one of the MQIASY_* values), but the value of the *ItemIndex* parameter is not MQIND_NONE. Only one instance of each system selector can exist in the bag.

Completion Code

MQCC_FAILED

Programmer response

Specify MQIND_NONE for the *ItemIndex* parameter.

2302 (08FE) (RC2302): MQRC_SYSTEM_ITEM_NOT_ALTERABLE:

Explanation

A call was issued to modify the value of a system data item in a bag (a data item with one of the MQIASY_* selectors), but the call failed because the data item is one that cannot be altered by the application.

Completion Code

MQCC_FAILED

Programmer response

Specify the selector of a user-defined data item, or remove the call.

2303 (08FF) (RC2303): MQRC_BAG_CONVERSION_ERROR:

Explanation

The mqBufferToBag or mqGetBag call was issued, but the data in the buffer or message could not be converted into a bag. This occurs when the data to be converted is not valid PCF.

Completion Code

MQCC_FAILED

Programmer response

Check the logic of the application that created the buffer or message to ensure that the buffer or message contains valid PCF.

If the message contains PCF that is not valid, the message cannot be retrieved using the mqGetBag call:
- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

2304 (0900) (RC2304): MQRC_SELECTOR_OUT_OF_RANGE:

Explanation

The *Selector* parameter has a value that is outside the valid range for the call. If the bag was created with the MQCBO_CHECK_SELECTORS option:

- For the mqAddInteger call, the value must be within the range MQIA_FIRST through MQIA_LAST.
- For the mqAddString call, the value must be within the range MQCA_FIRST through MQCA_LAST.

If the bag was not created with the MQCBO_CHECK_SELECTORS option:

- The value must be zero or greater.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value.

2305 (0901) (RC2305): MQRC_SELECTOR_NOT_UNIQUE:

Explanation

The *ItemIndex* parameter has the value MQIND_NONE, but the bag contains more than one data item with the selector value specified by the *Selector* parameter. MQIND_NONE requires that the bag contain only one occurrence of the specified selector.

This reason code also occurs on the mqExecute call when the administration bag contains two or more occurrences of a selector for a required parameter that permits only one occurrence.

Completion Code

MQCC_FAILED

Programmer response

Check the logic of the application that created the bag. If correct, specify for *ItemIndex* a value that is zero or greater, and add application logic to process all of the occurrences of the selector in the bag.

Review the description of the administration command being issued, and ensure that all required parameters are defined correctly in the bag.

2306 (0902) (RC2306): MQRC_INDEX_NOT_PRESENT:

Explanation

The specified index is not present:
- For a bag, this means that the bag contains one or more data items that have the selector value specified by the *Selector* parameter, but none of them has the index value specified by the *ItemIndex* parameter. The data item identified by the *Selector* and *ItemIndex* parameters must exist in the bag.
- For a namelist, this means that the index parameter value is too large, and outside the range of valid values.

Completion Code

MQCC_FAILED

Programmer response

Specify the index of a data item that does exist in the bag or namelist. Use the mqCountItems call to determine the number of data items with the specified selector that exist in the bag, or the nameCount method to determine the number of names in the namelist.

2307 (0903) (RC2307): MQRC_STRING_ERROR:

Explanation

The *String* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2308 (0904) (RC2308): MQRC_ENCODING_NOT_SUPPORTED:

Explanation

The *Encoding* field in the message descriptor MQMD contains a value that is not supported:
- For the mqPutBag call, the field in error resides in the *MsgDesc* parameter of the call.
- For the mqGetBag call, the field in error resides in:
  - The *MsgDesc* parameter of the call if the MQGMO_CONVERT option was specified.
  - The message descriptor of the message about to be retrieved if MQGMO_CONVERT was *not* specified.

Completion Code

MQCC_FAILED

Programmer response

The value must be MQENC_NATIVE.

If the value of the *Encoding* field in the message is not valid, the message cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

2309 (0905) (RC2309): MQRC_SELECTOR_NOT_PRESENT:

Explanation

The *Selector* parameter specifies a selector that does not exist in the bag.

Completion Code

MQCC_FAILED

Programmer response

Specify a selector that does exist in the bag.

2310 (0906) (RC2310): MQRC_OUT_SELECTOR_ERROR:

Explanation

The *OutSelector* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2311 (0907) (RC2311): MQRC_STRING_TRUNCATED:

Explanation

The string returned by the call is too long to fit in the buffer provided. The string has been truncated to fit in the buffer.

Completion Code

MQCC_FAILED

Programmer response

If the entire string is required, provide a larger buffer. On the mqInquireString call, the *StringLength* parameter is set by the call to indicate the size of the buffer required to accommodate the string without truncation.

2312 (0908) (RC2312): MQRC_SELECTOR_WRONG_TYPE:

Explanation

A data item with the specified selector exists in the bag, but has a data type that conflicts with the data type implied by the call being used. For example, the data item might have an integer data type, but the call being used might be mqSetString, which implies a character data type.

This reason code also occurs on the mqBagToBuffer, mqExecute, and mqPutBag calls when mqAddString or mqSetString was used to add the MQIACF_INQUIRY data item to the bag.

Completion Code

MQCC_FAILED

Programmer response

For the mqSetInteger and mqSetString calls, specify MQIND_ALL for the *ItemIndex* parameter to delete from the bag all existing occurrences of the specified selector before creating the new occurrence with the required data type.

For the mqInquireBag, mqInquireInteger, and mqInquireString calls, use the mqInquireItemInfo call to determine the data type of the item with the specified selector, and then use the appropriate call to determine the value of the data item.

For the mqBagToBuffer, mqExecute, and mqPutBag calls, ensure that the MQIACF_INQUIRY data item is added to the bag using the mqAddInteger or mqSetInteger calls.

2313 (0909) (RC2313): MQRC_INCONSISTENT_ITEM_TYPE:

Explanation

The mqAddInteger or mqAddString call was issued to add another occurrence of the specified selector to the bag, but the data type of this occurrence differed from the data type of the first occurrence.

This reason can also occur on the mqBufferToBag and mqGetBag calls, where it indicates that the PCF in the buffer or message contains a selector that occurs more than once but with inconsistent data types.

Completion Code

MQCC_FAILED

Programmer response

For the mqAddInteger and mqAddString calls, use the call appropriate to the data type of the first occurrence of that selector in the bag.

For the mqBufferToBag and mqGetBag calls, check the logic of the application that created the buffer or sent the message to ensure that multiple-occurrence selectors occur with only one data type. A message that contains a mixture of data types for a selector cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

2314 (090A) (RC2314): MQRC_INDEX_ERROR:

Explanation

An index parameter to a call or method has a value that is not valid. The value must be zero or greater. For bag calls, certain MQIND_* values can also be specified:

- For the mqDeleteItem, mqSetInteger and mqSetString calls, MQIND_ALL and MQIND_NONE are valid.
- For the mqInquireBag, mqInquireInteger, mqInquireString, and mqInquireItemInfo calls, MQIND_NONE is valid.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value.

2315 (090B) (RC2315): MQRC_SYSTEM_BAG_NOT_ALTERABLE:

Explanation

A call was issued to add a data item to a bag, modify the value of an existing data item in a bag, or retrieve a message into a bag, but the call failed because the bag is one that had been created by the system as a result of a previous mqExecute call. System bags cannot be modified by the application.

Completion Code

MQCC_FAILED

Programmer response

Specify the handle of a bag created by the application, or remove the call.

2316 (090C) (RC2316): MQRC_ITEM_COUNT_ERROR:

Explanation

The mqTruncateBag call was issued, but the *ItemCount* parameter specifies a value that is not valid. The value is either less than zero, or greater than the number of user-defined data items in the bag.

This reason also occurs on the mqCountItems call if the parameter pointer is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value. Use the mqCountItems call to determine the number of user-defined data items in the bag.

2317 (090D) (RC2317): MQRC_FORMAT_NOT_SUPPORTED:

Explanation

The *Format* field in the message descriptor MQMD contains a value that is not supported:

- In an administration message, the format value must be one of the following: MQFMT_ADMIN, MQFMT_EVENT, MQFMT_PCF. For the mqPutBag call, the field in error resides in the *MsgDesc* parameter of the call. For the mqGetBag call, the field in error resides in the message descriptor of the message about to be retrieved.
- On z/OS, the message was put to the command input queue with a format value of MQFMT_ADMIN, but the version of MQ being used does not support that format for commands.

Completion Code

MQCC_FAILED

Programmer response

If the error occurred when putting a message, correct the format value.

If the error occurred when getting a message, the message cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

2318 (090E) (RC2318): MQRC_SELECTOR_NOT_SUPPORTED:

Explanation

The *Selector* parameter specifies a value that is a system selector (a value that is negative), but the system selector is not one that is supported by the call.

Completion Code

MQCC_FAILED

Programmer response

Specify a selector value that is supported.

2319 (090F) (RC2319): MQRC_ITEM_VALUE_ERROR:

Explanation

The mqInquireBag or mqInquireInteger call was issued, but the `ItemValue` parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2320 (0910) (RC2320): MQRC_HBAG_ERROR:

Explanation

A call was issued that has a parameter that is a bag handle, but the handle is not valid. For output parameters, this reason also occurs if the parameter pointer is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2321 (0911) (RC2321): MQRC_PARAMETER_MISSING:

Explanation

An administration message requires a parameter that is not present in the administration bag. This reason code occurs only for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options.

Completion Code

MQCC_FAILED

Programmer response

Review the description of the administration command being issued, and ensure that all required parameters are present in the bag.

2322 (0912) (RC2322): MQRC_CMD_SERVER_NOT_AVAILABLE:

Explanation

The command server that processes administration commands is not available.

Completion Code

MQCC_FAILED

Programmer response

Start the command server.

2323 (0913) (RC2323): MQRC_STRING_LENGTH_ERROR:

Explanation

The *StringLength* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2324 (0914) (RC2324): MQRC_INQUIRY_COMMAND_ERROR:

Explanation

The mqAddInquiry call was used previously to add attribute selectors to the bag, but the command code to be used for the mqBagToBuffer, mqExecute, or mqPutBag call is not recognized. As a result, the correct PCF message cannot be generated.

Completion Code

MQCC_FAILED

Programmer response

Remove the mqAddInquiry calls and use instead the mqAddInteger call with the appropriate MQIACF_*_ATTRS or MQIACH_*_ATTRS selectors.

2325 (0915) (RC2325): MQRC_NESTED_BAG_NOT_SUPPORTED:

Explanation

A bag that is input to the call contains nested bags. Nested bags are supported only for bags that are output from the call.

Completion Code

MQCC_FAILED

Programmer response

Use a different bag as input to the call.

2326 (0916) (RC2326): MQRC_BAG_WRONG_TYPE:

Explanation

The *Bag* parameter specifies the handle of a bag that has the wrong type for the call. The bag must be an administration bag, that is, it must be created with the MQCBO_ADMIN_BAG option specified on the mqCreateBag call.

Completion Code

MQCC_FAILED

Programmer response

Specify the MQCBO_ADMIN_BAG option when the bag is created.

2327 (0917) (RC2327): MQRC_ITEM_TYPE_ERROR:

Explanation

The mqInquireItemInfo call was issued, but the *ItemType* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2328 (0918) (RC2328): MQRC_SYSTEM_BAG_NOT_DELETABLE:

Explanation

An mqDeleteBag call was issued to delete a bag, but the call failed because the bag is one that had been created by the system as a result of a previous mqExecute call. System bags cannot be deleted by the application.

Completion Code

MQCC_FAILED

Programmer response

Specify the handle of a bag created by the application, or remove the call.

2329 (0919) (RC2329): MQRC_SYSTEM_ITEM_NOT_DELETABLE:

Explanation

A call was issued to delete a system data item from a bag (a data item with one of the MQIASY_* selectors), but the call failed because the data item is one that cannot be deleted by the application.

Completion Code

MQCC_FAILED

Programmer response

Specify the selector of a user-defined data item, or remove the call.

2330 (091A) (RC2330): MQRC_CODED_CHAR_SET_ID_ERROR:

Explanation

The *CodedCharSetId* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2331 (091B) (RC2331): MQRC_MSG_TOKEN_ERROR:

Explanation

An MQGET call was issued to retrieve a message using the message token as a selection criterion, but the options specified are not valid, because MQMO_MATCH_MSG_TOKEN was specified with either MQGMO_WAIT or MQGMO_SET_SIGNAL.

An Async Consumer was registered to retrieve a message using the message token as a selection criterion, but when the delivery of messages for this consumer was started no message matching the message token was available for delivery to the consumer. As a result the consumer is suspended.

Completion Code

MQCC_FAILED

Programmer response

If this reason code is returned from an MQGET call, either remove the MQMO_MATCH_MSG_TOKEN match option, or remove the MQGMO_WAIT, or MQGMO_SET_SIGNAL option which was specified.

If this reason code is returned to an Async Consume Event Handler, then the consumer has been suspended and no further messages will be delivered to the consumer. The consumer should be de-registered or modified to select a different message using the MQCB call.

2332 (091C) (RC2332): MQRC_MISSING_WIH:

Explanation

An MQPUT or MQPUT1 call was issued to put a message on a queue with an `IndexType` attribute that had the value MQIT_MSG_TOKEN, but the `Format` field in the MQMD was not MQFMT_WORK_INFO_HEADER. This error occurs only when the message arrives at the destination queue manager.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Modify the application to ensure that it places an MQWIH structure at the start of the message data, and sets the `Format` field in the MQMD to MQFMT_WORK_INFO_HEADER. Alternatively, change the `ApplType` attribute of the process definition used by the destination queue to be MQAT_WLM, and specify the required service name and service step name in its `EnvData` attribute.

2333 (091D) (RC2333): MQRC_WIH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQWIH structure that is not valid. Possible errors include the following:

- The *StrucId* field is not MQWIH_STRUC_ID.
- The *Version* field is not MQWIH_VERSION_1.
- The *StrucLength* field is not MQWIH_LENGTH_1.
- The *CodedCharSetId* field is zero, or a negative value that is not valid.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).
- On z/OS, this error also occurs when the *IndexType* attribute of the queue is MQIT_MSG_TOKEN, but the message data does not begin with an MQWIH structure.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the *CodedCharSetId* field to a valid value (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in this field).

- On z/OS, if the queue has an *IndexType* of MQIT_MSG_TOKEN, ensure that the message data begins with an MQWIH structure.

2334 (091E) (RC2334): MQRC_RFH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQRFH or MQRFH2 structure that is not valid. Possible errors include the following:

- The *StrucId* field is not MQRFH_STRUC_ID.
- The *Version* field is not MQRFH_VERSION_1 (MQRFH), or MQRFH_VERSION_2 (MQRFH2).
- The *StrucLength* field specifies a value that is too small to include the structure plus the variable-length data at the end of the structure.
- The *CodedCharSetId* field is zero, or a negative value that is not valid.
- The *BufferLength* parameter of the call has a value that is too small to accommodate the structure (the structure extends beyond the end of the message).

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the *CodedCharSetId* field to a valid value (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in this field).

2335 (091F) (RC2335): MQRC_RFH_STRING_ERROR:

Explanation

The contents of the *NameValueString* field in the MQRFH structure are not valid. *NameValueString* must adhere to the following rules:

- The string must consist of zero or more name/value pairs separated from each other by one or more blanks; the blanks are not significant.
- If a name or value contains blanks that are significant, the name or value must be enclosed in double quotation marks.
- If a name or value itself contains one or more double quotation marks, the name or value must be enclosed in double quotation marks, and each embedded double quotation mark must be doubled.
- A name or value can contain any characters other than the null, which acts as a delimiter. The null and characters following it, up to the defined length of *NameValueString*, are ignored.

The following is a valid *NameValueString*:

```
Famous_Words "The program displayed ""Hello World"""
```

Completion Code

MQCC_FAILED

Programmer response

Modify the application that generated the message to ensure that it places in the *NameValueString* field data that adheres to the rules. Check that the *StrucLength* field is set to the correct value.

2336 (0920) (RC2336): MQRC_RFH_COMMAND_ERROR:

Explanation

The message contains an MQRFH structure, but the command name contained in the *NameValueString* field is not valid.

Completion Code

MQCC_FAILED

Programmer response

Modify the application that generated the message to ensure that it places in the *NameValueString* field a command name that is valid.

2337 (0921) (RC2337): MQRC_RFH_PARM_ERROR:

Explanation

The message contains an MQRFH structure, but a parameter name contained in the *NameValueString* field is not valid for the command specified.

Completion Code

MQCC_FAILED

Programmer response

Modify the application that generated the message to ensure that it places in the *NameValueString* field only parameters that are valid for the specified command.

2338 (0922) (RC2338): MQRC_RFH_DUPLICATE_PARM:

Explanation

The message contains an MQRFH structure, but a parameter occurs more than once in the *NameValueString* field when only one occurrence is valid for the specified command.

Completion Code

MQCC_FAILED

Programmer response

Modify the application that generated the message to ensure that it places in the *NameValueString* field only one occurrence of the parameter.

2339 (0923) (RC2339): MQRC_RFH_PARM_MISSING:

Explanation

The message contains an MQRFH structure, but the command specified in the *NameValueString* field requires a parameter that is not present.

Completion Code

MQCC_FAILED

Programmer response

Modify the application that generated the message to ensure that it places in the *NameValueString* field all parameters that are required for the specified command.

2340 (0924) (RC2340): MQRC_CHAR_CONVERSION_ERROR:

Explanation

This reason code is returned by the Java MQQueueManager constructor when a required character-set conversion is not available. The conversion required is between two nonUnicode character sets.

This reason code occurs in the following environment: MQ Classes for Java on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the National Language Resources component of the z/OS Language Environment is installed, and that conversion between the IBM-1047 and ISO8859-1 character sets is available.

2341 (0925) (RC2341): MQRC_UCS2_CONVERSION_ERROR:

Explanation

This reason code is returned by the Java MQQueueManager constructor when a required character set conversion is not available. The conversion required is between the UCS-2 Unicode character set and the character set of the queue manager which defaults to IBM-500 if no specific value is available.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the relevant Unicode conversion tables are available for the JVM. For z/OS ensure that the Unicode conversion tables are available to the z/OS Language Environment. The conversion tables should be installed as part of the z/OS C/C++ optional feature. Refer to the *z/OS C/C++ Programming Guide* for more information about enabling UCS-2 conversions.

2342 (0926) (RC2342): MQRC_DB2_NOT_AVAILABLE:

Explanation

An MQOPEN, MQPUT1, or MQSET call, or a command, was issued to access a shared queue, but it failed because the queue manager is not connected to a Db2 subsystem. As a result, the queue manager is unable to access the object definition relating to the shared queue.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Configure the Db2 subsystem so that the queue manager can connect to it.

2343 (0927) (RC2343): MQRC_OBJECT_NOT_UNIQUE:

Explanation

An MQOPEN or MQPUT1 call, or a command, was issued to access a queue, but the call failed because the queue specified cannot be resolved unambiguously. There exists a shared queue with the specified name, and a nonshared queue with the same name.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

One of the queues must be deleted. If the queue to be deleted contains messages, use the MQSC command MOVE QLOCAL to move the messages to a different queue, and then use the command DELETE QLOCAL to delete the queue.

2344 (0928) (RC2344): MQRC_CONN_TAG_NOT_RELEASED:

Explanation

An MQDISC call was issued when there was a unit of work outstanding for the connection handle. For CICS, IMS, and RRS connections, the MQDISC call does not commit or back out the unit of work. As a result, the connection tag associated with the unit of work is not yet available for reuse. The tag becomes available for reuse only when processing of the unit of work has been completed.

This reason code occurs only on z/OS.

Completion Code

MQCC_WARNING

Programmer response

Do not try to reuse the connection tag immediately. If the MQCONNX call is issued with the same connection tag, and that tag is still in use, the call fails with reason code MQRC_CONN_TAG_IN_USE.

2345 (0929) (RC2345): MQRC_CF_NOT_AVAILABLE:

Explanation

An MQI call was issued to access a shared queue, but the call failed either because connectivity was lost to the coupling facility (CF) where the CF structure specified in the queue definition was allocated, or because allocation of the CF structure failed because there is no suitable CF to hold the structure, based on the preference list in the active CFRM policy.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

If connectivity was lost to the CF where the structure was allocated, and the queue manager has been configured to tolerate the failure and rebuild the structure, no action should be necessary. Otherwise, make available a coupling facility with one of the names specified in the CFRM policy, or modify the CFRM policy to specify the names of coupling facilities that are available.

2346 (092A) (RC2346): MQRC_CF_STRUC_IN_USE:

Explanation

An MQI call or command was issued to operate on a shared queue, but the call failed because the coupling-facility structure specified in the queue definition is unavailable. The coupling-facility structure can be unavailable because a structure dump is in progress, or new connectors to the structure are currently inhibited, or an existing connector to the structure failed or disconnected abnormally and clean-up is not yet complete.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Typically, this is a temporary problem: wait for a while then retry the operation.

If the problem does not resolve itself, then connectivity problems experienced during the recovery of structures in the coupling facility could have occurred. In this case, restart the queue manager which reported the error. Resolve all the connectivity problems concerning the coupling facility before restarting the queue manager.

2347 (092B) (RC2347): MQRC_CF_STRUC_LIST_HDR_IN_USE:

Explanation

An MQGET, MQOPEN, MQPUT1, or MQSET call was issued to access a shared queue, but the call failed because the list header associated with the coupling-facility structure specified in the queue definition is temporarily unavailable. The list header is unavailable because it is undergoing recovery processing.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

The problem is temporary; wait a short while and then retry the operation.

2348 (092C) (RC2348): MQRC_CF_STRUC_AUTH_FAILED:

Explanation

An MQOPEN or MQPUT1 call was issued to access a shared queue, but the call failed because the user is not authorized to access the coupling-facility structure specified in the queue definition.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Modify the security profile for the user identifier used by the application so that the application can access the coupling-facility structure specified in the queue definition.

2349 (092D) (RC2349): MQRC_CF_STRUC_ERROR:

Explanation

An MQOPEN or MQPUT1 call was issued to access a shared queue, but the call failed because the coupling-facility structure name specified in the queue definition is not defined in the CFRM data set, or is not the name of a list structure.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Modify the queue definition to specify the name of a coupling-facility list structure that is defined in the CFRM data set.

2350 (092E) (RC2350): MQRC_CONN_TAG_NOT_USABLE:

Explanation

An MQCONNX call was issued specifying one of the MQCNO_*_CONN_TAG_* options, but the call failed because the connection tag specified by *ConnTag* in MQCNO is being used by the queue manager for recovery processing, and this processing is delayed pending recovery of the coupling facility.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

The problem is likely to persist. Consult the system programmer to ascertain the cause of the problem.

2351 (092F) (RC2351): MQRC_GLOBAL_UOW_CONFLICT:

Explanation

An attempt was made to use inside a global unit of work a connection handle that is participating in another global unit of work. This can occur when an application passes connection handles between objects where the objects are involved in different DTC transactions. Because transaction completion is asynchronous, it is possible for this error to occur *after* the application has finalized the first object and committed its transaction.

This error does not occur for nontransactional MQI calls.

This reason code occurs only on Windows and z/OS.

Completion Code

MQCC_FAILED

Programmer response

Check that the "MTS Transaction Support⌂ attribute defined for the object's class is set correctly. If necessary, modify the application so that the connection handle is not used by objects participating in different units of work.

2352 (0930) (RC2352): MQRC_LOCAL_UOW_CONFLICT:

Explanation

An attempt was made to use inside a global unit of work a connection handle that is participating in a queue-manager coordinated local unit of work. This can occur when an application passes connection handles between objects where one object is involved in a DTC transaction and the other is not.

This error does not occur for nontransactional MQI calls.

This reason code occurs only on Windows and z/OS.

Completion Code

MQCC_FAILED

Programmer response

Check that the "MTS Transaction Support⌂ attribute defined for the object's class is set correctly. If necessary, modify the application so that the connection handle is not used by objects participating in different units of work.

2353 (0931) (RC2353): MQRC_HANDLE_IN_USE_FOR_UOW:

Explanation

An attempt was made to use outside a unit of work a connection handle that is participating in a global unit of work.

This error can occur when an application passes connection handles between objects where one object is involved in a DTC transaction and the other is not. Because transaction completion is asynchronous, it is possible for this error to occur *after* the application has finalized the first object and committed its transaction.

This error can also occur when a single object that was created and associated with the transaction loses that association whilst the object is running. The association is lost when DTC terminates the transaction independently of MTS. This might be because the transaction timed out, or because DTC shut down.

This error does not occur for nontransactional MQI calls.

This reason code occurs only on Windows.

Completion Code

MQCC_FAILED

Programmer response

Check that the "MTS Transaction Support" attribute defined for the object's class is set correctly. If necessary, modify the application so that objects executing within different units of work do not try to use the same connection handle.

2354 (0932) (RC2354): MQRC_UOW_ENLISTMENT_ERROR:

Explanation

This reason code can occur for various reasons and occurs only on Windows , and HP Integrity NonStop Server .

On Windows, the most likely reason is that an object created by a DTC transaction does not issue a transactional MQI call until after the DTC transaction timed out. (If the DTC transaction times out after a transactional MQI call has been issued, reason code MQRC_HANDLE_IN_USE_FOR_UOW is returned by the failing MQI call.)

On HP Integrity NonStop Server, this reason occurs:

- On a transactional MQI call when the client encounters a configuration error preventing it from enlisting with the TMF/Gateway, therefore preventing participation within a global unit of work that is coordinated by the Transaction Management Facility (TMF).
- If a client application makes an enlistment request before the TMF/Gateway completes recovery of in-doubt transactions, the request is held for up to 1 second. If recovery does not complete within that time, the enlistment is rejected.

Another cause of MQRC_UOW_ENLISTMENT_ERROR is incorrect installation; On Windows, for example, the Windows NT Service pack must be installed after the Windows NT Option pack.

Completion Code

MQCC_FAILED

Programmer response

On Windows, check the DTC "Transaction timeout" value. If necessary, verify the Windows NT installation order.

On HP Integrity NonStop Server this might be a configuration error. The client issues a message to the client error log providing extra information about the configuration error. Contact your system administrator to resolve the indicated error.

2355 (0933) (RC2355): MQRC_UOW_MIX_NOT_SUPPORTED:

Explanation

This reason code occurs only on Windows when you are running a version of the queue manager before version 5.2. , and on HP Integrity NonStop Server .

On Windows, the following explanations might apply:
- The mixture of calls that is used by the application to perform operations within a unit of work is not supported. In particular, it is not possible to mix within the same process a local unit of work that is coordinated by the queue manager with a global unit of work that is coordinated by DTC (Distributed Transaction Coordinator).
- An application might cause this mixture to arise if some objects in a package are coordinated by DTC and others are not. It can also occur if transactional MQI calls from an MTS client are mixed with transactional MQI calls from a library package transactional MTS object.
- No problem arises if all transactional MQI calls originate from transactional MTS objects, or all transactional MQI calls originate from non-transactional MTS objects. But when a mixture of styles is used, the first style that is used fixes the style for the unit of work, and subsequent attempts to use the other style within the process fail with reason code MQRC_UOW_MIX_NOT_SUPPORTED.
- When an application is run twice, scheduling factors in the operating system mean that it is possible for the queue-manager-coordinated transactional calls to fail in one run, and for the DTC-coordinated transactional calls to fail in the other run.

On HP Integrity NonStop Server it is not possible, within a single IBM MQ connection, to issue transactional MQI calls under the coordination of the Transaction Management Facility (TMF) if transactional MQI calls have already been made within a local unit of work that is coordinated by the queue manager until the local unit of work is completed by issuing either MQCMIT or MQBACK.

Completion Code

MQCC_FAILED

Programmer response

On Windows, check that the "MTS Transaction Support" attribute defined for the object's class is set correctly. If necessary, modify the application so that objects that run within different units of work do not try to use the same connection handle.

On HP Integrity NonStop Server, if a local unit of work that is coordinated by the queue manager is in progress, it must either be completed by issuing MQCMIT, or rolled back by issuing MQBACK before issuing any transactional MQI calls under the coordination of TMF.

2356 (0934) (RC2356): MQRC_WXP_ERROR:

Explanation

An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the workload exit parameter structure *ExitParms* is not valid, for one of the following reasons:

- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The *StrucId* field is not MQWXP_STRUC_ID.
- The *Version* field is not MQWXP_VERSION_2.
- The *CacheContext* field does not contain the value passed to the exit by the queue manager.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the parameter specified for *ExitParms* is the MQWXP structure that was passed to the exit when the exit was invoked.

2357 (0935) (RC2357): MQRC_CURRENT_RECORD_ERROR:

Explanation

An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the address specified by the *CurrentRecord* parameter is not the address of a valid record. *CurrentRecord* must be the address of a destination record (MQWDR), queue record (MQWQR), or cluster record (MQWCR) residing within the cluster cache.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the cluster workload exit passes the address of a valid record residing in the cluster cache.

2358 (0936) (RC2358): MQRC_NEXT_OFFSET_ERROR:

Explanation

An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the offset specified by the *NextOffset* parameter is not valid. *NextOffset* must be the value of one of the following fields:

- *ChannelDefOffset* field in MQWDR
- *ClusterRecOffset* field in MQWDR
- *ClusterRecOffset* field in MQWQR
- *ClusterRecOffset* field in MQWCR

Completion Code

MQCC_FAILED

Programmer response

Ensure that the value specified for the *NextOffset* parameter is the value of one of the fields listed.

2359 (0937) (RC2359): MQRC_NO_RECORD_AVAILABLE:

Explanation

An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the current record is the last record in the chain.

Completion Code

MQCC_FAILED

Programmer response

None.

2360 (0938) (RC2360): MQRC_OBJECT_LEVEL_INCOMPATIBLE:

Explanation

An MQOPEN or MQPUT1 call, or a command, was issued, but the definition of the object to be accessed is not compatible with the queue manager to which the application has connected. The object definition was created or modified by a different version of the queue manager.

If the object to be accessed is a queue, the incompatible object definition could be the object specified, or one of the object definitions used to resolve the specified object (for example, the base queue to which an alias queue resolves, or the transmission queue to which a remote queue or queue-manager alias resolves).

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

The application must be run on a queue manager that is compatible with the object definition. .

2361 (0939) (RC2361): MQRC_NEXT_RECORD_ERROR:

Explanation

An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the address specified for the *NextRecord* parameter is either null, not valid, or the address of read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Specify a valid address for the *NextRecord* parameter.

2362 (093A) (RC2362): MQRC_BACKOUT_THRESHOLD_REACHED:

Explanation

This reason code occurs only in the *Reason* field in an MQDLH structure, or in the *Feedback* field in the MQMD of a report message.

A JMS ConnectionConsumer found a message that exceeds the queue's backout threshold. The queue does not have a backout requeue queue defined, so the message was processed as specified by the disposition options in the *Report* field in the MQMD of the message.

On queue managers that do not support the *BackoutThreshold* and *BackoutRequeueQName* queue attributes, JMS ConnectionConsumer uses a value of 20 for the backout threshold. When the *BackoutCount* of a message reaches this threshold, the message is processed as specified by the disposition options.

If the *Report* field specifies one of the MQRO_EXCEPTION_* options, this reason code appears in the *Feedback* field of the report message. If the *Report* field specifies MQRO_DEAD_LETTER_Q, or the disposition report options remain at the default, this reason code appears in the *Reason* field of the MQDLH.

Completion Code

None

Programmer response

Investigate the cause of the backout count being greater than the threshold. To correct this, define the backout queue for the queue concerned.

2363 (093B) (RC2363): MQRC_MSG_NOT_MATCHED:

Explanation

This reason code occurs only in the `Reason` field in an MQDLH structure, or in the `Feedback` field in the MQMD of a report message.

While performing Point-to-Point messaging, JMS encountered a message matching none of the selectors of ConnectionConsumers monitoring the queue. To maintain performance, the message was processed as specified by the disposition options in the `Report` field in the MQMD of the message.

If the `Report` field specifies one of the MQRO_EXCEPTION_* options, this reason code appears in the `Feedback` field of the report message. If the `Report` field specifies MQRO_DEAD_LETTER_Q, or the disposition report options remain at the default, this reason code appears in the `Reason` field of the MQDLH.

Completion Code

None

Programmer response

To correct this, ensure that the ConnectionConsumers monitoring the queue provide a complete set of selectors. Alternatively, set the QueueConnectionFactory to retain messages.

2364 (093C) (RC2364): MQRC_JMS_FORMAT_ERROR:

Explanation

This reason code is generated by JMS applications that use either:
- ConnectionConsumers
- Activation Specifications
- WebSphere Application Server Listener Ports

and connect to an IBM MQ queue manager using IBM MQ messaging provider migration mode. When the IBM MQ classes for JMS encounter a message that cannot be parsed (for example, the message contains an invalid RFH2 header) the message is processed as specified by the disposition options in the `Report` field in the MQMD of the message.

If the `Report` field specifies one of the MQRO_EXCEPTION_* options, this reason code appears in the `Feedback` field of the report message. If the `Report` field specifies MQRO_DEAD_LETTER_Q, or the disposition report options remain at the default, this reason code appears in the `Reason` field of the MQDLH.

Completion Code

None

Programmer response

Investigate the origin of the message.

2365 (093D) (RC2365): MQRC_SEGMENTS_NOT_SUPPORTED:

Explanation

An MQPUT call was issued to put a segment of a logical message, but the queue on which the message is to be placed has an *IndexType* of MQIT_GROUP_ID. Message segments cannot be placed on queues with this index type.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Modify the application to put messages that are not segments; ensure that the MQMF_SEGMENT and MQMF_LAST_SEGMENT flags in the *MsgFlags* field in MQMD are not set, and that the *Offset* is zero. Alternatively, change the index type of the queue.

2366 (093E) (RC2366): MQRC_WRONG_CF_LEVEL:

Explanation

An MQOPEN or MQPUT1 call was issued specifying a shared queue, but the queue requires a coupling-facility structure with a different level of capability.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the coupling-facility structure used for the queue is at the level required to support the capabilities that the queue provides.

You can use the DISPLAY CFSTRUCT command to display the level, and ALTER CFSTRUCT() CFLEVEL() command to modify the level; see The MQSC commands.

2367 (093F) (RC2367): MQRC_CONFIG_CREATE_OBJECT:

Explanation

This condition is detected when an object is created.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Create object.

2368 (0940) (RC2368): MQRC_CONFIG_CHANGE_OBJECT:

Explanation

This condition is detected when an object is changed.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Change object.

2369 (0941) (RC2369): MQRC_CONFIG_DELETE_OBJECT:

Explanation

This condition is detected when an object is deleted.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Delete object.

2370 (0942) (RC2370): MQRC_CONFIG_REFRESH_OBJECT:

Explanation

This condition is detected when an object is refreshed.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Refresh object.

2371 (0943) (RC2371): MQRC_CHANNEL_SSL_ERROR:

Explanation

This condition is detected when a connection cannot be established due to an SSL key-exchange or authentication failure.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel SSL Error.

2373 (0945) (RC2373): MQRC_CF_STRUC_FAILED:

Explanation

An MQI call or command was issued to access a shared queue, but the call failed because the coupling-facility structure used for the shared queue had failed.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Report the problem to the operator or administrator, who should use the MQSC command RECOVER CFSTRUCT to initiate recovery of the coupling-facility structure, unless automatic recovery has been enabled for the structure.

2374 (0946) (RC2374): MQRC_API_EXIT_ERROR:

Explanation

An API exit function returned an invalid response code, or failed in some other way.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Check the exit logic to ensure that the exit is returning valid values in the *ExitResponse* and *ExitResponse2* fields of the MQAXP structure. Consult the FFST record to see if it contains more detail about the problem.

2375 (0947) (RC2375): MQRC_API_EXIT_INIT_ERROR:

Explanation

The queue manager encountered an error while attempting to initialize the execution environment for an API exit function.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Consult the FFST record to obtain more detail about the problem.

2376 (0948) (RC2376): MQRC_API_EXIT_TERM_ERROR:

Explanation

The queue manager encountered an error while attempting to terminate the execution environment for an API exit function.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Consult the FFST record to obtain more detail about the problem.

2377 (0949) (RC2377): MQRC_EXIT_REASON_ERROR:

Explanation

An MQXEP call was issued by an API exit function, but the value specified for the `ExitReason` parameter is either not valid, or not supported for the specified function identifier `Function`.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Modify the exit function to specify a value for `ExitReason` that is valid for the specified value of `Function`.

2378 (094A) (RC2378): MQRC_RESERVED_VALUE_ERROR:

Explanation

An MQXEP call was issued by an API exit function, but the value specified for the `Reserved` parameter is not valid. The value must be the null pointer.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Modify the exit to specify the null pointer as the value of the `Reserved` parameter.

2379 (094B) (RC2379): MQRC_NO_DATA_AVAILABLE:

Explanation

This reason should be returned by the MQZ_ENUMERATE_AUTHORITY_DATA installable service component when there is no more authority data to return to the invoker of the service component.
- On z/OS, this reason code does not occur.

Completion Code

MQCC_FAILED

Programmer response

None.

2380 (094C) (RC2380): MQRC_SCO_ERROR:

Explanation

On an MQCONNX call, the MQSCO structure is not valid for one of the following reasons:
- The *StrucId* field is not MQSCO_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Correct the definition of the MQSCO structure.

2381 (094D) (RC2381): MQRC_KEY_REPOSITORY_ERROR:

Explanation

On an MQCONN or MQCONNX call, the location of the key repository is either not specified, not valid, or results in an error when used to access the key repository. The location of the key repository is specified by one of the following:
- The value of the MQSSLKEYR environment variable (MQCONN or MQCONNX call), or
- The value of the *KeyRepository* field in the MQSCO structure (MQCONNX call only).

For the MQCONNX call, if both MQSSLKEYR and *KeyRepository* are specified, the latter is used.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid location for the key repository.

2382 (094E) (RC2382): MQRC_CRYPTO_HARDWARE_ERROR:

Explanation

On an MQCONN or MQCONNX call, the configuration string for the cryptographic hardware is not valid, or results in an error when used to configure the cryptographic hardware. The configuration string is specified by one of the following:
- The value of the MQSSLCRYP environment variable (MQCONN or MQCONNX call), or
- The value of the *CryptoHardware* field in the MQSCO structure (MQCONNX call only).

For the MQCONNX call, if both MQSSLCRYP and *CryptoHardware* are specified, the latter is used.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid configuration string for the cryptographic hardware.

2383 (094F) (RC2383): MQRC_AUTH_INFO_REC_COUNT_ERROR:

Explanation

On an MQCONNX call, the *AuthInfoRecCount* field in the MQSCO structure specifies a value that is less than zero.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Specify a value for *AuthInfoRecCount* that is zero or greater.

2384 (0950) (RC2384): MQRC_AUTH_INFO_REC_ERROR:

Explanation

On an MQCONNX call, the MQSCO structure does not specify the address of the MQAIR records correctly. One of the following applies:
- *AuthInfoRecCount* is greater than zero, but *AuthInfoRecOffset* is zero and *AuthInfoRecPtr* is the null pointer.
- *AuthInfoRecOffset* is not zero and *AuthInfoRecPtr* is not the null pointer.
- *AuthInfoRecPtr* is not a valid pointer.
- *AuthInfoRecOffset* or *AuthInfoRecPtr* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Ensure that one of *AuthInfoRecOffset* or *AuthInfoRecPtr* is zero and the other nonzero. Ensure that the field used points to accessible storage.

2385 (0951) (RC2385): MQRC_AIR_ERROR:

Explanation

On an MQCONNX call, an MQAIR record is not valid for one of the following reasons:
- The *StrucId* field is not MQAIR_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Correct the definition of the MQAIR record.

2386 (0952) (RC2386): MQRC_AUTH_INFO_TYPE_ERROR:

Explanation

On an MQCONNX call, the *AuthInfoType* field in an MQAIR record specifies a value that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Specify MQAIT_CRL_LDAP for *AuthInfoType*.

2387 (0953) (RC2387): MQRC_AUTH_INFO_CONN_NAME_ERROR:

Explanation

On an MQCONNX call, the *AuthInfoConnName* field in an MQAIR record specifies a value that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid connection name.

2388 (0954) (RC2388): MQRC_LDAP_USER_NAME_ERROR:

Explanation

On an MQCONNX call, an LDAP user name in an MQAIR record is not specified correctly. One of the following applies:

- *LDAPUserNameLength* is greater than zero, but *LDAPUserNameOffset* is zero and *LDAPUserNamePtr* is the null pointer.
- *LDAPUserNameOffset* is nonzero and *LDAPUserNamePtr* is not the null pointer.
- *LDAPUserNamePtr* is not a valid pointer.
- *LDAPUserNameOffset* or *LDAPUserNamePtr* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Ensure that one of *LDAPUserNameOffset* or *LDAPUserNamePtr* is zero and the other nonzero. Ensure that the field used points to accessible storage.

2389 (0955) (RC2389): MQRC_LDAP_USER_NAME_LENGTH_ERR:

Explanation

On an MQCONNX call, the *LDAPUserNameLength* field in an MQAIR record specifies a value that is less than zero.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Specify a value for *LDAPUserNameLength* that is zero or greater.

2390 (0956) (RC2390): MQRC_LDAP_PASSWORD_ERROR:

Explanation

On an MQCONNX call, the *LDAPPassword* field in an MQAIR record specifies a value when no value is allowed.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Specify a value that is blank or null.

2391 (0957) (RC2391): MQRC_SSL_ALREADY_INITIALIZED:

Explanation

An MQCONN or MQCONNX call was issued when a connection is already open to the same queue manager. There is a conflict between the SSL options of the connections for one of three reasons:
- The SSL configuration options are different between the first and second connections.
- The existing connection was specified without SSL configuration options, but the second connection has SSL configuration options specified.
- The existing connection was specified with SSL configuration options, but the second connection does not have any SSL configuration options specified.

The connection to the queue manager completed successfully, but the SSL configuration options specified on the call were ignored; the existing SSL environment was used instead.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_WARNING

Programmer response

If the application must be run with the SSL configuration options defined on the MQCONN or MQCONNX call, use the MQDISC call to sever the connection to the queue manager and then stop the application. Alternatively run the application later when the SSL environment has not been initialized.

2392 (0958) (RC2392): MQRC_SSL_CONFIG_ERROR:

Explanation

On an MQCONNX call, the MQCNO structure does not specify the MQSCO structure correctly. One of the following applies:
- *SSLConfigOffset* is nonzero and *SSLConfigPtr* is not the null pointer.
- *SSLConfigPtr* is not a valid pointer.
- *SSLConfigOffset* or *SSLConfigPtr* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Ensure that one of *SSLConfigOffset* or *SSLConfigPtr* is zero and the other nonzero. Ensure that the field used points to accessible storage.

2393 (0959) (RC2393): MQRC_SSL_INITIALIZATION_ERROR:

Explanation

An MQCONN or MQCONNX call was issued with SSL configuration options specified, but an error occurred during the initialization of the SSL environment.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_FAILED

Programmer response

Check that the SSL installation is correct.

2394 (095A) (RC2394): MQRC_Q_INDEX_TYPE_ERROR:

Explanation

An MQGET call was issued specifying one or more of the following options:
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOGICAL_ORDER

but the call failed because the queue is not indexed by group identifier. These options require the queue to have an `IndexType` of MQIT_GROUP_ID.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Redefine the queue to have an `IndexType` of MQIT_GROUP_ID. Alternatively, modify the application to avoid using the options listed.

2395 (095B) (RC2395): MQRC_CFBS_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFBS structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2396 (095C) (RC2396): MQRC_SSL_NOT_ALLOWED:

Explanation

A connection to a queue manager was requested, specifying SSL encryption. However, the connection mode requested is one that does not support SSL (for example, bindings connect).

Completion Code

MQCC_FAILED

Programmer response

Modify the application to request client connection mode, or to disable SSL encryption.

Note: Using a non null setting, including blanks, for the connection's cipher suite property can also cause this error.

2397 (095D) (RC2397): MQRC_JSSE_ERROR:

Explanation

JSSE reported an error (for example, while connecting to a queue manager using SSL encryption). The MQException object containing this reason code references the Exception thrown by JSSE; this can be obtained by using the MQException.getCause() method. From JMS, the MQException is linked to the thrown JMSException.

This reason code occurs only with Java applications.

Completion Code

MQCC_FAILED

Programmer response

Inspect the causal exception to determine the JSSE error.

2398 (095E) (RC2398): MQRC_SSL_PEER_NAME_MISMATCH:

Explanation

The application attempted to connect to the queue manager using SSL encryption, but the distinguished name presented by the queue manager does not match the specified pattern.

Completion Code

MQCC_FAILED

Programmer response

Check the certificates used to identify the queue manager. Also check the value of the sslPeerName property specified by the application.

2399 (095F) (RC2399): MQRC_SSL_PEER_NAME_ERROR:

Explanation

The application specified a peer name of incorrect format.

Completion Code

MQCC_FAILED

Programmer response

Check the value of the sslPeerName property specified by the application.

2400 (0960) (RC2400): MQRC_UNSUPPORTED_CIPHER_SUITE:

Explanation

A connection to a queue manager was requested, specifying SSL encryption. However, JSSE reported that it does not support the CipherSuite specified by the application.

This reason code occurs only with Java applications.

Completion Code

MQCC_FAILED

Programmer response

Check the CipherSuite specified by the application. Note that the names of JSSE CipherSuites differ from their equivalent CipherSpecs used by the queue manager.

Also, check that JSSE is correctly installed.

2401 (0961) (RC2401): MQRC_SSL_CERTIFICATE_REVOKED:

Explanation

A connection to a queue manager was requested, specifying SSL encryption. However, the certificate presented by the queue manager was found to be revoked by one of the specified CertStores.

This reason code occurs only with Java applications.

Completion Code

MQCC_FAILED

Programmer response

Check the certificates used to identify the queue manager.

2402 (0962) (RC2402): MQRC_SSL_CERT_STORE_ERROR:

Explanation

A connection to a queue manager was requested, specifying SSL encryption. However, none of the CertStore objects provided by the application could be searched for the certificate presented by the queue manager. The MQException object containing this reason code references the Exception encountered when searching the first CertStore; this can be obtained using the MQException.getCause() method. From JMS, the MQException is linked to the thrown JMSException.

This reason code occurs only with Java applications.

Completion Code

MQCC_FAILED

Programmer response

Inspect the causal exception to determine the underlying error. Check the CertStore objects provided by your application. If the causal exception is a java.lang.NoSuchElementException, ensure that your application is not specifying an empty collection of CertStore objects.

2406 (0966) (RC2406): MQRC_CLIENT_EXIT_LOAD_ERROR:

Explanation

The external user exit required for a client connection could not be loaded because the shared library specified for it cannot be found, or the entry point specified for it cannot be found.

This reason code occurs only with Java applications.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct library has been specified, and that the path variable for the machine environment includes the relevant directory. Ensure also that the entry point has been named properly and that the named library does export it.

2407 (0967) (RC2407): MQRC_CLIENT_EXIT_ERROR:

Explanation

A failure occurred while executing a non-Java user exit for a client connection.

This reason code occurs only with Java applications.

Completion Code

MQCC_FAILED

Programmer response

Check that the non-Java user exit can accept the parameters and message being passed to it and that it can handle error conditions, and that any information that the exit requires, such as user data, is correct and available.

2409 (0969) (RC2409): MQRC_SSL_KEY_RESET_ERROR:

Explanation

On an MQCONN or MQCONNX call, the value of the SSL key reset count is not in the valid range of 0 through 999 999 999.

The value of the SSL key reset count is specified by either the value of the MQSSLRESET environment variable (MQCONN or MQCONNX call), or the value of the *KeyResetCount* field in the MQSCO structure (MQCONNX call only). For the MQCONNX call, if both MQSSLRESET and *KeyResetCount* are specified, the latter is used. MQCONN or MQCONNX

If you specify an SSL/TLS secret key reset count in the range 1 byte through 32Kb, SSL/TLS channels will use a secret key reset count of 32Kb. This is to avoid the overhead of excessive key resets which would occur for small SSL/TLS secret key reset values.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure and the MQSSLRESET environment variable are set correctly.

2411 (096B) (RC2411): MQRC_LOGGER_STATUS:

Explanation

This condition is detected when a logger event occurs.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Logger.

2412 (096C) (RC2412): MQRC_COMMAND_MQSC:

Explanation

This condition is detected when an MQSC command is executed.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Command.

2413 (096D) (RC2413): MQRC_COMMAND_PCF:

Explanation

This condition is detected when a PCF command is executed.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Command.

2414 (096E) (RC2414): MQRC_CFIF_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFIF structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2415 (096F) (RC2415): MQRC_CFSF_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFSF structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2416 (0970) (RC2416): MQRC_CFGR_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFGR structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2417 (0971) (RC2417): MQRC_MSG_NOT_ALLOWED_IN_GROUP:

An explanation of the error, completion code, and programmer response.

Explanation

An MQPUT or MQPUT1 call was issued to put a message in a group but it is not valid to put such a message in a group. An example of an invalid message is a PCF message where the Type is MQCFT_TRACE_ROUTE.

You cannot use grouped or segmented messages with Publish/Subscribe.

Completion Code

MQCC_FAILED

Programmer response

Remove the invalid message from the group.

2418 (0972) (RC2418): MQRC_FILTER_OPERATOR_ERROR:

Explanation

The **Operator** parameter supplied is not valid.

If it is an input variable then the value is not one of the MQCFOP_* constant values. If it is an output variable then the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredicatable results occur.)

Completion Code

MQCC_FAILED

Programmer response

Correct the parameter.

2419 (0973) (RC2419): MQRC_NESTED_SELECTOR_ERROR:

Explanation

An mqAddBag call was issued, but the bag to be nested contained a data item with an inconsistent selector. This reason only occurs if the bag into which the nested bag was to be added was created with the MQCBO_CHECK_SELECTORS option.

Completion Code

MQCC_FAILED

Programmer response

Ensure that all data items within the bag to be nested have selectors that are consistent with the data type implied by the item.

2420 (0974) (RC2420): MQRC_EPH_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQEPH structure that is not valid. Possible errors include the following:
- The `StrucId` field is not MQEPH_STRUC_ID.
- The `Version` field is not MQEPH_VERSION_1.
- The `StrucLength` field specifies a value that is too small to include the structure plus the variable-length data at the end of the structure.
- The `CodedCharSetId` field is zero, or a negative value that is not valid.
- The `Flags` field contains an invalid combination of MQEPH_* values.
- The `BufferLength` parameter of the call has a value that is too small to accommodate the structure, so the structure extends beyond the end of the message.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly. Ensure that the application sets the `CodedCharSetId` field to a valid value; note that MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are not valid in this field.

2421 (0975) (RC2421): MQRC_RFH_FORMAT_ERROR:

Explanation

The message contains an MQRFH structure, but its format is incorrect. If you are using IBM MQ SOAP, the error is in an incoming SOAP/MQ request message.

Completion Code

MQCC_FAILED

Programmer response

If you are using IBM MQ SOAP with the IBM-supplied sender, contact your IBM support center. If you are using IBM MQ SOAP with a bespoke sender, check that the RFH2 section of the SOAP/MQ request message is in valid RFH2 format.

2422 (0976) (RC2422): MQRC_CFBF_ERROR:

Explanation

An MQPUT or MQPUT1 call was issued, but the message data contains an MQCFBF structure that is not valid.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus IBM MQ clients connected to these systems.

Completion Code

MQCC_FAILED

Programmer response

Check that the fields in the structure are set correctly.

2423 (0977) (RC2423): MQRC_CLIENT_CHANNEL_CONFLICT:

Explanation

A client channel definition table (CCDT) was specified for determining the name of the channel, but the name has already been defined.

This reason code occurs only with Java applications.

Completion Code

MQCC_FAILED

Programmer response

Change the channel name to blank and try again.

2424 (0978) (RC2424): MQRC_SD_ERROR:

Explanation

On the MQSUB call, the Subscription Descriptor MQSD is not valid, for one of the following reasons:
- The StrucId field is not MQSD_SCTRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid (it is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results can occur).
- The queue manager cannot copy the changes structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQSD structure are set correctly.

2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR:

Explanation

On the MQOPEN or MQPUT1 call in the Object Descriptor MQOD, or on the MQSUB call in the Subscription Descriptor MQSD the resultant full topic string is not valid.

One of the following applies:
- ObjectName contains the name of a TOPIC object with a TOPICSTR attribute that contains an empty topic string.
- The fully resolved topic string contains the escape character `'%'` and it is not followed by one of the characters, `'*', '?' or '%'`, and the MQSO_WILDCARD_CHAR option has been used on an MQSUB call.
- On an MQOPEN, conversion cannot be performed using the CCSID specified in the MQOD structure.
- The topic string is greater than 255 characters when using IBM MQ Multicast messaging.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that there are no invalid topic string characters in either ObjectString or ObjectName.

If using IBM MQ Multicast messaging, ensure that the topic string is less than 255 characters.

2426 (097A) (RC2426): MQRC_STS_ERROR:

Explanation

On an MQSTAT call, the MQSTS structure is not valid, for one of the following reasons:
- The StrucId field is not MQSTS_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQSTS structure are set correctly.

2428 (097C) (RC2428): MQRC_NO_SUBSCRIPTION:

Explanation

An MQSUB call using option MQSO_RESUME was made specifying a full subscription name that does not match any existing subscription.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the subscription exists and that the full subscription name is correctly specified in your application. The full subscription name is built from the ConnTag field specified at connection time in the MQCNO structure and the SubName field specified at MQSUB time in the MQSD structure.

2429 (097D) (RC2429): MQRC_SUBSCRIPTION_IN_USE:

Explanation

An MQSUB call using option MQSO_RESUME was made specifying a full subscription name that is in use.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the subscription name is correctly specified in your application. The subscription name is specified in the SubName field in the MQSD structure.

2430 (097E) (RC2430): MQRC_STAT_TYPE_ERROR:

Explanation

The STS parameter contains options that are not valid for the MQSTAT call. This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Programmer response

Specify a valid MQSTS structure as a parameter on the call to MQSTAT.

2431 (097F) (RC2431): MQRC_SUB_USER_DATA_ERROR:

Explanation

On the MQSUB call in the Subscription Descriptor MQSD the SubUserData field is not valid. One of the following applies:

- SubUserData.VSLength is greater than zero, but SubUserData.VSOffset is zero and SubUserData.VSPtr is the null pointer.
- SubUserData.VSOffset is nonzero and SubUserData.VSPtr is not the null pointer (that is, it appears both fields are being used where only one is allowed).
- SubUserData.VSPtr is not a valid pointer.
- SubUserData.VSOffset or SubUserData.VSPtr points to storage that is not accessible.
- SubUserData.VSLength exceeds the maximum length allowed for this field.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that one of SubUserData.VSOffset or SubUserData.VSPtr is zero and the other nonzero. Ensure that the field used points to accessible storage. Specify a length that does not exceed the maximum length allowed for this field.

2432 (0980) (RC2432): MQRC_SUB_ALREADY_EXISTS:

Explanation

An MQSUB call was issued to create a subscription, using the MQSO_CREATE option, but a subscription using the same SubName and ObjectString already exists.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the SubName and ObjectString input fields in the MQSD structure are set correctly, or use the MQSO_RESUME option to get a handle for the subscription that already exists.

2434 (0982) (RC2434): MQRC_IDENTITY_MISMATCH:

Explanation

An MQSUB call using either MQSO_RESUME or MQSO_ALTER was made against a subscription that has the MQSO_FIXED_USERID option set, by a userid other than the one recorded as owning the subscription.

Completion Code

MQCC_FAILED

Programmer Response

Correct the full subscription name to one that is unique, or update the existing subscription to allow different userids to use it by using the MQSO_ANY_USERID option from an application running under the owning userid.

2435 (0983) (RC2435): MQRC_ALTER_SUB_ERROR:
Explanation

An MQSUB call using option MQSO_ALTER was made changing a subscription that was created with the MQSO_IMMUTABLE option.

Completion Code

MQCC_FAILED

Programmer Response

Remove the subscription using MQCLOSE and re-create it with MQSUB with the attributes set correctly.

2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED:

Explanation

An MQSUB call using the MQSO_DURABLE option failed. This can be for one of the following reasons:
- The topic subscribed to is defined as DURSUB(NO).
- The queue named SYSTEM.DURABLE.SUBSCRIBER.QUEUE is not available.
- The topic subscribed to is defined as both MCAST(ONLY) and DURSUB(YES) (or DURSUB(ASPARENT) and the parent is DURSUB(YES)).

Completion Code

MQCC_FAILED

Programmer Response

Durable subscriptions are stored on the SYSTEM.DURABLE.SUBSCRIBER.QUEUE. Ensure that this queue is available for use. Possible reasons for failure include the queue being full, the queue being put inhibited, the queue not existing, or (on z/OS) the pageset the queue is defined to use doesn't exist.

If the topic subscribed to is defined as DURSUB(NO) either alter the administrative topic node to use DURSUB(YES) or use the MQSO_NON_DURABLE option instead.

If the topic subscribed to is defined as MCAST(ONLY) when using IBM MQ Multicast messaging, alter the topic to use DURSUB(NO).

2437 (0985) (RC2437): MQRC_NO_RETAINED_MSG:

Explanation

An MQSUBRQ call was made to a topic to request that any retained publications for this topic are sent to the subscriber. However, there are no retained publications currently stored for this topic.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that publishers to the topic are marking their publication to be retained and that publications are being made to this topic.

2438 (0986) (RC2438): MQRC_SRO_ERROR:

Explanation

On the MQSUBRQ call, the Subscription Request Options MQSRO is not valid, for one of the following reasons:
- The StrucId field is not MQSRO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQSRO structure are set correctly.

2440 (0988) (RC2440): MQRC_SUB_NAME_ERROR:

Explanation

On the MQSUB call in the Subscription Descriptor MQSD the SubName field is not valid or has been omitted. This is required if the MQSD option MQSO_DURABLE is specified, but may also be used if MQSO_DURABLE is not specified.

One of the following applies:
- SubName.VSLength is greater than zero, but SubName.VSOffset is zero and SubName.VSPtr is the null pointer.
- SubName.VSOffset is nonzero and SubName.VSPtr is not the null pointer (that is, it appears both fields are being used where only one is allowed).
- SubName.VSPtr is not a valid pointer.

- SubName.VSOffset or SubName.VSPtr points to storage that is not accessible.
- SubName.VSLength is zero but this field is required.
- SubName.VSLength exceeds the maximum length allowed for this field.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that SubName is specified and SubName.VSLength is nonzero. Ensure that one of SubName.VSOffset or SubName.VSPtr is zero and the other nonzero. Ensure that the field used points to accessible storage. Specify a length that does not exceed the maximum length allowed for this field.

This code can be returned if the sd.Options flags MQSO_CREATE and MQSO_RESUME are set together and sd.SubName is not initialized. You must also initialize the MQCHARV structure for sd.SubName, even if there is no subscription to resume; see Example 2: Managed MQ subscriber for more details.

2441 (0989) (RC2441): MQRC_OBJECT_STRING_ERROR:

Explanation

On the MQOPEN or MQPUT1 call in the Object Descriptor MQOD, or on the MQSUB call in the Subscription Descriptor MQSD the ObjectString field is not valid.

One of the following applies:
- ObjectString.VSLength is greater than zero, but ObjectString.VSOffset is zero and ObjectString.VSPtr is the null pointer.
- ObjectString.VSOffset is nonzero and ObjectString.VSPtr is not the null pointer (that is, it appears both fields are being used where only one is allowed).
- ObjectString.VSPtr is not a valid pointer.
- ObjectString.VSOffset or ObjectString.VSPtr points to storage that is not accessible.
- ObjectString.VSLength exceeds the maximum length allowed for this field.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that one of ObjectString.VSOffset or ObjectString.VSPtr is zero and the other nonzero. Ensure that the field used points to accessible storage. Specify a length that does not exceed the maximum length allowed for this field.

2442 (098A) (RC2442): MQRC_PROPERTY_NAME_ERROR:

Explanation

An attempt was made to set a property with an invalid name. Using any of the following settings results in this error:

- The name contains an invalid character.
- The name begins "JMS" or "usr.JMS" and the JMS property is not recognized.
- The name begins "mq" in any mixture of lowercase or uppercase and is not "mq_usr" and contains more than one "." character (U+002E). Multiple "." characters are not allowed in properties with those prefixes.
- The name is "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" and "ESCAPE" or is one of these keywords prefixed by "usr.".
- The name begins with "Body" or "Root" (except for names beginning "Root.MQMD.").
- A "." character must not be followed immediately by another "." character.
- The "." character cannot be the last character in a property name.

Completion Code

MQCC_FAILED

Programmer Response

Valid property names are described in the IBM MQ documentation. Ensure that all properties in the message have valid names before reissuing the call.

2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED:

Explanation

An MQPUT or MQPUT1 call was issued to put a segmented message or a message that may be broken up into smaller segments (MQMF_SEGMENTATION_ALLOWED). The message was found to contain one or more MQ-defined properties in the message data; MQ-defined properties are not valid in the message data of a segmented message.

IBM MQ Multicast cannot use segmented messages.

Completion Code

MQCC_FAILED

Programmer Response

Remove the invalid properties from the message data or prevent the message from being segmented.

2444 (098C) (RC2444): MQRC_CBD_ERROR:

Explanation

a MQCB call the MQCBD structure is not valid for one of the following reasons:
- The StrucId field is not MQCBD_STRUC_ID
- The Version field is specifies a value that is not valid or is not supported
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQCBD structure are set correctly.

2445 (098D) (RC2445): MQRC_CTLO_ERROR:

Explanation

On a MQCTL call the MQCTLO structure is not valid for one of the following reasons:
- The StrucId field is not MQCTLO_STRUC_ID
- The Version field is specifies a value that is not valid or is not supported
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQCTLO structure are set correctly.

2446 (098E) (RC2446): MQRC_NO_CALLBACKS_ACTIVE:

Explanation

An MQCTL call was made with an Operation of MQOP_START_WAIT and has returned because there are no currently defined callbacks which are not suspended.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that there is at least one registered, resumed consumer function.

2448 (0990) (RC2448): MQRC_CALLBACK_NOT_REGISTERED:

Explanation

An attempt to issue an MQCB call has been made against an object handle which does not currently have a registered callback.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that a callback has been registered against the object handle.

2449 (0991) (RC2449): MQRC_OPERATION_NOT_ALLOWED:

Explanation

An MQCTL call was made with an Operation that is not allowed because of the state of asynchronous consumption on the hConn is currently in.

If Operation was MQOP_RESUME, the operation is not allowed because the state of asynchronous consumption on the hConn is STOPPED. Re-issue MQCTL with the MQOP_START Operation.

If Operation was MQOP_SUSPEND, the operation is not allowed because the state of asynchronous consumption on the hConn is STOPPED. If you need to get your hConn into a SUSPENDED state, issue MQCTL with the MQOP_START Operation followed by MQCTL with MQOP_SUSPEND.

If Operation was MQOP_START, the operation is not allowed because the state of asynchronous consumption on the hConn is SUSPENDED. Re-issue MQCTL with the MQOP_RESUME Operation.

If Operation was MQOP_START_WAIT, the operation is not allowed because either
- The state of asynchronous consumption on the hConn is SUSPENDED. Re-issue MQCTL with the MQOP_RESUME Operation.
- The state of asynchronous consumption on the hConn is already STARTED. Do not mix the use of MQOP_START and MQOP_START_WAIT within one application.

Completion Code

MQCC_FAILED

Programmer Response

Re-issue the MQCTL call with the correct Operation.

2457 (0999) (RC2457): MQRC_OPTIONS_CHANGED:

Explanation

An MQGET call on a queue handle opened using MQOO_READ_AHEAD (or resolved to that value through the queue's default value) has altered an option that is required to be consistent between MQGET calls.

Completion Code

MQCC_FAILED

Programmer Response

Keep all required MQGET options the same between invocations of MQGET, or use MQOO_NO_READ_AHEAD when opening the queue.

2458 (099A) (RC2458): MQRC_READ_AHEAD_MSGS:

Explanation

On an MQCLOSE call, the option MQCO_QUIESCE was used and there are still messages stored in client read ahead buffer that were sent to the client ahead of an application requesting them and have not yet been consumed by the application.

Completion Code

MQCC_WARNING

Programmer Response

Continue to consume messages using the queue handle until there are no more available and then issue the MQCLOSE again, or choose to discard these messages by issuing the MQCLOSE call with the MQCO_IMMEDIATE option instead.

2459 (099B) (RC2459): MQRC_SELECTOR_SYNTAX_ERROR:

Explanation

An MQOPEN, MQPUT1 or MQSUB call was issued but a selection string was specified which contained a syntax error.

Completion Code

MQCC_FAILED

Programmer Response

See Message selector syntax and ensure that you have correctly followed the rules for specifying selection strings. Correct any syntax errors and resubmit the MQ API call for which the error occurred.

2460 (099C) (RC2460): MQRC_HMSG_ERROR:

Explanation

On an MQCRTMH, MQDLTMH, MQSETMP, MQINQMP or MQDLT call, a message handle supplied is not valid, for one of the following reasons:

- The parameter pointer is not valid, or (for the MQCRTMH call) points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The value specified was not returned by a preceding MQCRTMH call.
- The value specified has been made invalid by a preceding MQDLTMH call.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that a successful MQCRTMH call is performed for the connection, and that an MQDLTMH call has not already been performed for it. Ensure that the handle is being used within its valid scope, for more information, see MQCRTMH - Create message handle.

2461 (099D) (RC2461): MQRC_CMHO_ERROR:

Explanation

On an MQCRTMH call, the create message handle options structure MQCMHO is not valid, for one of the following reasons:

- The StrucId field is not MQCMHO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQCMHO structure are set correctly.

2462 (099E) (RC2462): MQRC_DMHO_ERROR:

Explanation

On an MQDLTMH call, the delete message handle options structure MQDMHO is not valid, for one of the following reasons:
- The StrucId field is not MQCMHO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQDMHO structure are set correctly.

2463 (099F) (RC2463): MQRC_SMPO_ERROR:

Explanation

On an MQSETMP call, the set message property options structure MQSMPO is not valid, for one of the following reasons:
- The StrucId field is not MQSMPO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQSMPO structure are set correctly.

2464 (09A0) (RC2464): MQRC_IMPO_ERROR:

Explanation

On an MQINQMP call, the inquire message property options structure MQIMPO is not valid, for one of the following reasons:
- The StrucId field is not MQIMPO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQIMPO structure are set correctly.

2465 (09A1) (RC2465): MQRC_PROPERTY_NAME_TOO_BIG:

Explanation

On an MQINQMP call, IBM MQ attempted to copy the name of the inquired property into the location indicated by the ReturnedName field of the InqPropOpts parameter but the buffer was too small to contain the full property name. The call failed but the VSLength field of the ReturnedName of the InqPropOpts parameter indicates how large the ReturnedName buffer needs to be.

Completion Code

MQCC_FAILED

Programmer response

The full property name can be retrieved by calling MQINQMP again with a larger buffer for the returned name, also specifying the MQIMPO_INQ_PROP_UNDER_CURSOR option. This will inquire on the same property.

2466 (09A2) (RC2466): MQRC_PROP_VALUE_NOT_CONVERTED:

Explanation

An MQINQMP call was issued with the MQIMPO_CONVERT_VALUE option specified in the InqPropOpts parameter, but an error occurred during conversion of the value of the property. The property value is returned unconverted, the values of the ReturnedCCSID and ReturnedEncoding fields in the InqPropOpts parameter are set to those of the value returned.

Completion Code

MQCC_FAILED

Programmer Response

Check that the property value is correctly described by the ValueCCSID and ValueEncoding parameters that were specified when the property was set. Also check that these values, and the RequestedCCSID and RequestedEncoding specified in the InqPropOpts parameter of the MQINQMP call, are supported for MQ conversion. If the required conversion is not supported, conversion must be carried out by the application.

2467 (09A3) (RC2467): MQRC_PROP_TYPE_NOT_SUPPORTED:

Explanation

An MQINQMP call was issued and the property inquired has an unsupported data type. A string representation of the value is returned and the TypeString field of the InqPropOpts parameter can be used to determine the data type of the property.

Completion Code

MQCC_WARNING

Programmer Response

Check whether the property value was intended to have a data type indicated by the TypeString field. If so the application must decide how to interpret the value. If not modify the application that set the property to give it a supported data type.

2469 (09A5) (RC2469): MQRC_PROPERTY_VALUE_TOO_BIG:

Explanation

On an MQINQMP call, the property value was too large to fit into the supplied buffer. The DataLength field is set to the length of the property value before truncation and the Value parameter contains as much of the value as fits.

On an MQMHBUF call, the BufferLength was less than the size of the properties to be put in the buffer. In this case the call fails. The DataLength field is set to the length of the properties before truncation.

Completion Code

MQCC_WARNING

MQCC_FAILED

Programmer Response

Supply a buffer that is at least as large as DataLength if all of the property value data is required and call MQINQMP again with the MQIMPO_INQ_PROP_UNDER_CURSOR option specified.

2470 (09A6) (RC2470): MQRC_PROP_CONV_NOT_SUPPORTED:

Explanation

On an MQINQMP call, the MQIMPO_CONVERT_TYPE option was specified to request that the property value be converted to the supplied data type before the call returned. Conversion between the actual and requested property data types is not supported. The Type parameter indicates the data type of the property value.

Completion Code

MQCC_FAILED

Programmer Response

Either call MQINQMP again without MQIMPO_CONVERT_TYPE specified, or request a data type for which conversion is supported.

2471 (09A7) (RC2471): MQRC_PROPERTY_NOT_AVAILABLE:

Explanation

On an MQINQMP call, no property could be found that matched the specified name. When iterating through multiple properties, possibly using a name containing a wildcard character, this indicates that all properties matching the name have now been returned.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the correct property name was specified. If the name contains a wildcard character specify option MQIMPO_INQ_FIRST to begin iterating over the properties again.

2472 (09A8) (RC2472): MQRC_PROP_NUMBER_FORMAT_ERROR:

Explanation

On an MQINQMP call, conversion of the property value was requested. The format of the property is invalid for conversion to the requested data type.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the correct property name and data type were specified. Ensure that the application setting the property gave it the correct format. See the documentation for the MQINQMP call for details on the formats required for data conversion of property values.

2473 (09A9) (RC2473): MQRC_PROPERTY_TYPE_ERROR:

Explanation

On an MQSETMP call, the Type parameter does not specify a valid MQTYPE_* value. For properties beginning "Root.MQMD." or "JMS" the specified Type must correspond to the data type of the matching MQMD or JMS header field:

- For MQCHARn or Java String fields use MQTYPE_STRING.
- For MQLONG or Java int fields use MQTYPE_INT32.
- For MQBYTEn fields use MQTYPE_BYTE_STRING.
- For Java long fields use MQTYPE_INT64.

On an MQINQMP call, the Type parameter is not valid. Either the parameter pointer is not valid, the value is invalid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Correct the parameter.

2478 (09AE) (RC2478): MQRC_PROPERTIES_TOO_BIG:

Explanation

An MQPUT or MQPUT1 call was issued to put a message on a queue, but the properties of the message were too large. The length of the properties cannot exceed the value of the **MaxPropertiesLength** queue manager attribute. This return code will also be issued if a message with headers greater than 511 KB is put to a shared queue.

Completion Code

MQCC_FAILED

Programmer Response

Consider one of the following actions:
- Reduce the number or the size of the properties associated with the message. This could include moving some of the properties into the application data.
- Increase the value of the MaxPropertiesLength queue manager attribute.

2479 (09AF) (RC2479): MQRC_PUT_NOT_RETAINED:

Explanation

An MQPUT or MQPUT1 call was issued to publish a message on a topic, using the MQPMO_RETAIN option, but the publication was unable to be retained. The publication is not published to any matching subscribers.

Completion Code

MQCC_FAILED

Programmer Response

Retained publications are stored on the SYSTEM.RETAINED.PUB.QUEUE. Ensure that this queue is available for use by the application. Possible reasons for failure include the queue being full, the queue being put inhibited, or the queue not existing.

2480 (09B0) (RC2480): MQRC_ALIAS_TARGTYPE_CHANGED:

Explanation

An MQPUT or MQPUT1 call was issed to publish a message on a topic. One of the subscriptions matching this topic was made with a destination queue that was an alias queue which originally referenced a queue, but now references a topic object, which is not allowed. In this situation the reason code MQRC_ALIAS_TARGTYPE_CHANGED is returned in the Feedback field in the MQMD of a report message, or in the Reason field in the MQDLH structure of a message on the dead-letter queue.

Completion Code

MQCC_FAILED

Programmer Response

Find the subscriber that is using an alias queue which references a topic object and change it to reference a queue again, or change the subscription to reference a different queue.

2481 (09B1) (RC2481): MQRC_DMPO_ERROR:

Explanation

On an MQDLTMP call, the delete message property options structure MQDMPO is not valid, for one of the following reasons:
- The StrucId field is not MQDMPO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQDMPO structure are set correctly.

2482 (09B2) (RC2482): MQRC_PD_ERROR:

Explanation

On an MQSETMP or MQINQMP call, the property descriptor structure MQPD is not valid, for one of the following reasons:
- The StrucId field is not MQPD_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The Context field contains an unrecognized value.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQPD structure are set correctly.

2483 (09B3) (RC2483): MQRC_CALLBACK_TYPE_ERROR:

Explanation

An MQCB call was made with an Operation of MQOP_REGISTER with an incorrect value for CallbackType

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the CallbackType field of the MQCBDO is specified correctly.

2484 (09B4) (RC2484): MQRC_CBD_OPTIONS_ERROR:

Explanation

An MQCB call was made with an Operation of MQOP_REGISTER with an incorrect value for the Options field of the MQCBD.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the Options are specified correctly.

2485 (09B5) (RC2485): MQRC_MAX_MSG_LENGTH_ERROR:

Explanation

An MQCB call was made with an Operation of MQOP_REGISTER with an incorrect value for the MaxMsgLength field of the MQCBD.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the MaxMsgLength are specified correctly.

2486 (09B6) (RC2486): MQRC_CALLBACK_ROUTINE_ERROR:

Explanation

An MQCB call was made with an Operation of MQOP_REGISTER failed for one of the following reasons:

- Both CallbackName and CallbackFunction are specified. Only one must be specified on the call.
- The call was made from an environment not supporting function pointers.
- A programming language that does not support Function pointer references.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the CallbackName value is specified correctly.

2487 (09B7) (RC2487): MQRC_CALLBACK_LINK_ERROR:

Explanation

On an MQCTL call, the callback handling module (CSQBMCSM or CSQBMCSX for batch and DFHMQMCM for CICS) could not be loaded, so the adapter could not link to it.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the correct library concatenation has been specified in the application program execution JCL, and in the queue-manager startup JCL. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

2488 (09B8) (RC2488): MQRC_OPERATION_ERROR:

Explanation

An MQCTL or MQCB call was made with an invalid parameter.

There is a conflict with the value specified for **`Operation`** parameter.

This error can be caused by an invalid value in the **`Operation`** parameter, no registered consumers when using MQOP_START or MQOP_START_WAIT parameter, and trying to use non-threaded libraries with asynchronous API calls.

Completion Code

MQCC_FAILED

Programmer Response

Investigate the application program and verify the **`Operation`** parameter options are correct. Ensure you have link edited the application with the correct version of the threading libraries for asynchronous functions.

2489 (09B9) (RC2489): MQRC_BMHO_ERROR:
Explanation

On an MQBUFMH call, the buffer to message handle options structure MQBMHO is not valid, for one of the following reasons:
- • The StrucId field is not MQBMHO_STRUC_ID.
- • The Version field specifies a value that is not valid or not supported.
- • The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQBMHO structure are set correctly.

2490 (09BA) (RC2490): MQRC_UNSUPPORTED_PROPERTY:

Explanation

A message was found to contain a property that the queue manager does not support. The operation that failed required all the properties to be supported by the queue manager. This can occur on the MQPUT/MQPUT1 call or when a message is about to be sent down a channel to a queue manager than does not support message properties.

Completion Code

MQCC_FAILED

Programmer Response

Determine which property of the message is not supported by the queue manager and decide whether to remove the property from the message or connect to a queue manager which does support the property.

2492 (09BC) (RC2492): MQRC_PROP_NAME_NOT_CONVERTED:

Explanation

An MQINQMP call was issued with the MQIMPO_CONVERT_VALUE option specified in the InqPropOpts parameter, but an error occurred during conversion of the returned name of the property. The returned name is unconverted

Completion Code

MQCC_WARNING

Programmer Response

Check that the character set of the returned name was correctly described when the property was set. Also check that these values, and the RequestedCCSID and RequestedEncoding specified in the InqPropOpts parameter of the MQINQMP call, are supported for MQ conversion. If the required conversion is not supported, conversion must be carried out by the application.

2494 (09BE) (RC2494): MQRC_GET_ENABLED:

Explanation

This reason code is returned to an asynchronous consumer at the time a queue that was previously inhibited for get has been re-enabled for get.

Completion Code

MQCC_WARNING

Programmer Response

None. This reason code is used to inform the application of the change in state of the queue.

2495 (09BF) (RC2495): MQRC_MODULE_NOT_FOUND:

Explanation

A native shared library could not be loaded.

Completion Code

MQCC_FAILED

Programmer Response

This problem could be caused by either of the two following reasons:

- A MQCB call was made with an Operation of MQOP_REGISTER specifying a *CallbackName* which could not be found. Ensure that the *CallbackName* value is specified correctly.
- The Java MQ code could not load a Java native shared library. This error can occur if a Java application is running in a 32-bit JRE but has been configured to load the 64-bit Java Native Libraries. Check the associated Exception stack and FFST. Ensure that the JNI shared library is specified correctly. Check also that you have specified `-Djava.library.path=/opt/mqm/java/lib`, or equivalent, when invoking the Java program.

Related information:

The Java Native Interface (JNI) libraries required by IBM MQ classes for JMS applications

2496 (09C0) (RC2496): MQRC_MODULE_INVALID:

Explanation

An MQCB call was made with an Operation of MQOP_REGISTER, specifying a CallbackName which is not a valid load module.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the CallbackName value is specified correctly.

2497 (09C1) (RC2497): MQRC_MODULE_ENTRY_NOT_FOUND:

Explanation

An MQCB call was made with an Operation of MQOP_REGISTER and the CallbackName identifies a function name which can't be found in the specified library.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the CallbackName value is specified correctly.

2498 (09C2) (RC2498): MQRC_MIXED_CONTENT_NOT_ALLOWED:

Explanation

An attempt was made to set a property with mixed content. For example, if an application set the property "x.y" and then attempted to set the property "x.y.z" it is unclear whether in the property name hierarchy "y" contains a value or another logical grouping. Such a hierarchy would be "mixed content" and this is not supported. Setting a property which would cause mixed content is not allowed. A hierarchy within a property name is created using the "." character (U+002E).

Completion Code

MQCC_FAILED

Programmer Response

Valid property names are described in the IBM MQ documentation. Change the property name hierarchy so that it no longer contains mixed content before re-issuing the call.

2499 (09C3) (RC2499): MQRC_MSG_HANDLE_IN_USE:

Explanation

A message property call was called (MQCRTMH, MQDLTMH, MQSETMP, MQINQMP, MQDLTMP or MQMHBUF) specifying a message handle that is already in use on another API call. A message handle may only be used on one call at a time.

Concurrent use of a message handle can arise, for example, when an application uses multiple threads.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the message handle cannot be used while another call is in progress.

2500 (09C4) (RC2500): MQRC_HCONN_ASYNC_ACTIVE:

Explanation

An attempt to issue an MQI call has been made while the connection is started.

Completion Code

MQCC_FAILED

Programmer Response

Stop or suspend the connection using the MQCTL call and retry the operation.

2501 (09C5) (RC2501): MQRC_MHBO_ERROR:

Explanation

On an MQMHBUF call, the message handle to buffer options structure MQMHBO is not valid, for one of the following reasons:
- The StrucId field is not MQMHBO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQMHBO structure are set correctly.

2502 (09C6) (RC2502): MQRC_PUBLICATION_FAILURE:

Explanation

An MQPUT or MQPUT1 call was issued to publish a message on a topic. Delivery of the publication to one of the subscribers failed and due to the combination of the syncpoint option used and either:

- • The PMSGDLV attribute on the administrative TOPIC object if it was a persistent message.
- • The NPMSGDLV attribute on the administrative TOPIC object if it was a non-persistent message.

The publication has not been delivered to any of the subscribers.

Completion Code

MQCC_FAILED

Programmer response

Find the subscriber or subscribers who are having problems with their subscription queue and resolve the problem, or change the setting of the PMSGDLV or NPMSGDLV attributes on the TOPIC so that problems with one subscriber do not have an effect on other subscribers. Retry the MQPUT.

2503 (09C7) (RC2503): MQRC_SUB_INHIBITED:

Explanation

MQSUB calls are currently inhibited for the topic subscribed to.

Completion Code

MQCC_FAILED

Programmer Response

If the system design allows subscription requests to be inhibited for short periods, retry the operation later.

2504 (09C8) (RC2504): MQRC_SELECTOR_ALWAYS_FALSE:

Explanation

An MQOPEN, MQPUT1 or MQSUB call was issued but a selection string was specified which will never select a message

Completion Code

MQCC_FAILED

Programmer Response

Verify that the logic of the selection string which was passed in on the API is as expected. Make any necessary corrections to the logic of the string and resubmit the MQ API call for which the message occurred.

2507 (09CB) (RC2507): MQRC_XEPO_ERROR:

Explanation

On an MQXEP call, the exit options structure MQXEPO is not valid, for one of the following reasons:
- The StrucId field is not MQXEPO_STRUC_ID.
- The Version field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code

MQCC_FAILED

Programmer Response

Ensure that input fields in the MQXEPO structure are set correctly.

2509 (09CD) (RC2509): MQRC_DURABILITY_NOT_ALTERABLE:
Explanation

An MQSUB call using option MQSO_ALTER was made changing the durability of the subscription. The durability of a subscription cannot be changed.

Completion Code

MQCC_FAILED

Programmer Response

Remove the subscription using MQCLOSE and re-create it with MQSUB with the attributes set correctly, or change the durability option used on the MQSUB call so that it matches the existing subscription.

2510 (09CE) (RC2510): MQRC_TOPIC_NOT_ALTERABLE:
Explanation

An MQSUB call using option MQSO_ALTER was made changing the one or more of the fields in the MQSD that provide the topic being subscribed to. These fields are the ObjectName, ObjectString, or wildcard options. The topic subscribed to cannot be changed.

Completion Code

MQCC_FAILED

Programmer Response

Remove the subscription using MQCLOSE and re-create it with MQSUB with the attributes set correctly, or change the attributes and options used on the MQSUB call so that it matches the existing subscription.

2512 (09D0) (RC2512): MQRC_SUBLEVEL_NOT_ALTERABLE:
Explanation

An MQSUB call using option MQSO_ALTER was made changing the SubLevel of the subscription. The SubLevel of a subscription cannot be changed.

Completion Code

MQCC_FAILED

Programmer Response

Remove the subscription using MQCLOSE and re-create it with MQSUB with the attributes set correctly, or change the SubLevel field used on the MQSUB call so that it matches the existing subscription.

2513 (09D1) (RC2513): MQRC_PROPERTY_NAME_LENGTH_ERR:

Explanation

An attempt was made to set, inquire or delete a property with an invalid name. This is for one of the following reasons:
- The VSLength field of the property name was set to less than or equal to zero.
- The VSLength field of the property name was set to greater than the maximum allowed value (see constant MQ_MAX_PROPERTY_NAME_LENGTH).
- The VSLength field of the property name was set to MQVS_NULL_TERMINATED and the property name was greater than the maximum allowed value.

Completion Code

MQCC_FAILED

Programmer Response

Valid property names are described in the IBM MQ documentation. Ensure that the property has a valid name length before issuing the call again.

2514 (09D2) (RC2514): MQRC_DUPLICATE_GROUP_SUB:

Explanation

An MQSUB call using option MQSO_GROUP_SUB was made creating a new grouped subscription but, although it has a unique SubName, it matches the Full topic name of an existing subscription in the group.

Completion Code

MQCC_FAILED

Programmer Response

Correct the Full topic name used so that it does not match any existing subscription in the group, or correct the grouping attributes if, either a different group was intended or the subscription was not intended to be grouped at all.

2515 (09D3) (RC2515): MQRC_GROUPING_NOT_ALTERABLE:
Explanation

An MQSUB call was made using option MQSO_ALTER on a grouped subscription, that is one made with the option MQSO_GROUP_SUB. Grouping of subscriptions is not alterable.

Completion Code

MQCC_FAILED

Programmer response

Remove the subscription using MQCLOSE and re-create it with MQSUB with the attributes set correctly, or change the various grouping fields used on the MQSUB call so that it matches the existing subscription.

2516 (09D4) (RC2516): MQRC_SELECTOR_INVALID_FOR_TYPE:

Explanation

A SelectionString may only be specified in the MQOD for an MQOPEN/MQPUT1 if the following is true:
- ObjectType is MQOT_Q
- The queue is being opened using one of the MQOO_INPUT_* open options.

Completion Code

MQCC_FAILED

Programmer Response

Modify the value of ObjectType to be MQOT_Q and ensure that the queue is being opened using one of the MQOO_INPUT_* options.

2517 (09D5) (RC2517): MQRC_HOBJ_QUIESCED:

Explanation

The HOBJ has been quiesced but there are no messages in the read ahead buffer which match the current selection criteria. This reason code indicates that the read ahead buffer is not empty.

Completion Code

MQCC_FAILED

Programmer Response

This reason code indicates that all messages with the current selection criteria have been processed. Do one of the following:
- If no further messages need to be processed issue an MQCLOSE without the MQCO_QUIESCE option. Any messages in the read ahead buffer will be discarded.
- Relax the current selection criteria by modifying the values in the MQGMO and reissue the call. Once all messages have been consumed the call will return MQRC_HOBJ_QUIESCED_NO_MSGS.

2518 (09D6) (RC2518): MQRC_HOBJ_QUIESCED_NO_MSGS:

Explanation

The HOBJ has been quiesced and the read ahead buffer is now empty. No further messages will be delivered to this HOBJ

Completion Code

MQCC_FAILED

Programmer Response

Issue MQCLOSE against the HOBJ.

2519 (09D7) (RC2519): MQRC_SELECTION_STRING_ERROR:
Explanation

The SelectionString must be specified according to the description of how to use an MQCHARV structure. Examples of why this error was returned:
- SelectionString.VSLength is greater than zero, but SelectionString.VSOffset is zero and SelectionString.VSPtr is a null pointer.
- SelectionString.VSOffset is nonzero and SelectionString.VSPtr is not the null pointer (that is, it appears both fields are being used where only one is allowed).
- SelectionString.VSPtr is not a valid pointer.
- SelectionString.VSOffset or SelectionString.VSPtr points to storage that is not accessible.
- SelectionString.VSLength exceeds the maximum length allowed for this field. The maximum length is determined by MQ_SELECTOR_LENGTH.

Completion Code

MQCC_FAILED

Programmer Response

Modify the fields of the MQCHARV so that it follows the rules for a valid MQCHARV structure.

2520 (09D8) (RC2520): MQRC_RES_OBJECT_STRING_ERROR:

Explanation

On the MQOPEN or MQPUT1 call in the Object Descriptor MQOD, or on the MQSUB call in the Subscription Descriptor MQSD the ResObjectString field is not valid.

One of the following applies:
- ResObjectString.VSLength is greater than zero, but ResObjectString.VSOffset is zero and ResObjectString.VSPtr is the null pointer.
- ResObjectString.VSOffset is nonzero and ResObjectString.VSPtr is not the null pointer (that is, it appears both fields are being used where only one is allowed).
- ResObjectString.VSPtr is not a valid pointer.
- ResObjectString.VSOffset or ResObjectString.VSPtr points to storage that is not accessible.
- ResObjectString.VSBufSize is MQVS_USE_VSLENGTH and one of ResObjectString.VSOffset or ResObjectString.VSPtr have been provided.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that one of ResObjectString.VSOffset or ResObjectString.VSPtr is zero and the other nonzero and that the buffer length is provided in ResObjectString.VSBufSize. Ensure that the field used points to accessible storage.

2521 (09D9) (RC2521): MQRC_CONNECTION_SUSPENDED:

Explanation

An MQCTL call with Operation MQOP_START_WAIT has returned because the asynchronous consumption of messages has been suspended. This can be for the following reasons:

- The connection was explicitly suspended using MQCTL with Operation MQOP_SUSPEND
- All consumers have been either unregistered or suspended.

Completion Code

MQCC_WARNING

Programmer Response

If this is an expected condition, no corrective action required. If this is an unexpected condition check that:

- At least one consumer is registered and not suspended
- The connection has not been suspended

2522 (09DA) (RC2522): MQRC_INVALID_DESTINATION:

Explanation

An MQSUB call failed because of a problem with the destination where publications messages are to be sent, so an object handle cannot be returned to the application and the subscription is not made. This can be for one of the following reasons:

- The MQSUB call used MQSO_CREATE, MQSO_MANAGED and MQSO_NON_DURABLE and the model queue referred to by MNDURMDL on the administrative topic node does not exist
- The MQSUB call used MQSO_CREATE, MQSO_MANAGED and MQSO_DURABLE and the model queue referred to by MDURMDL on the administrative topic node does not exist, or has been defined with a DEFTYPE of TEMPDYN.
- The MQSUB call used MQSO_CREATE or MQSO_ALTER on a durable subscription and the object handle provided referred to a temporary dynamic queue. This is not an appropriate destination for a durable subscription.
- The MQSUB call used MQSO_RESUME and a Hobj of MQHO_NONE, to resume an administratively created subscription, but the queue name provided in the DEST parameter of the subscription does not exist.
- The MQSUB call used MQSO_RESUME and a Hobj of MQHO_NONE, to resume a previously created API subscription, but the queue previously used no longer exists.

Completion Code

MQCC_FAILED

Programmer Response

Ensure that the model queues referred to by MNDURMDL and MDURMDL exist and have an appropriate DEFTYPE. Create the queue referred to by the DEST parameter in an administrative subscription if one is being used. Alter the subscription to use an existing queue if the previously used one does not exist.

2523 (09DB) (RC2523): MQRC_INVALID_SUBSCRIPTION:

Explanation

An MQSUB call using MQSO_RESUME or MQSO_ALTER failed because the subscription named is not valid for use by applications. This can be for one of the following reasons:
- The subscription is the SYSTEM.DEFAULT.SUB subscription which is not a valid subscription and should only be used to fill in the default values on DEFINE SUB commands.
- The subscription is a proxy type subscription which is not a valid subscription for an application to resume and is only used to enable publications to be forwarded between queue managers.
- The subscription has expired and is no longer valid for use.

Completion Code

MQCC_FAILED

Programmer Response

Ensure the subscription named in SubName field is not one of the invalid ones listed. If you have a handle open to the subscription already it must have expired. Use MQCLOSE to close the handle and then if necessary create a new subscription.

2524 (09DC) (RC2524): MQRC_SELECTOR_NOT_ALTERABLE:

Explanation

An MQSUB call was issued with the MQSO_ALTER option and the MQSD contained a SelectionString. It is not valid to alter the SelectionString of a subscription.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the SelectionString field of the MQSD does not contain a valid VSPtr and that the VSLength is set to zero when making a call to MQSUB.

2525 (09DD) (RC2525): MQRC_RETAINED_MSG_Q_ERROR:

Explanation

An MQSUB call which did not use the MQSO_NEW_PUBLICATIONS_ONLY option, or an MQSUBRQ call, failed because the retained publications which exist for the topic string subscribed to cannot be retrieved from the SYSTEM.RETAINED.PUB.QUEUE. This can be for one of the following reasons:

- The queue has become damaged or has been deleted.
- The queue has been set to GET(DISABLED).
- Messages have been removed from this queue directly.

An error message will be written to the log giving more details about the problem with the SYSTEM.RETAINED.PUB.QUEUE.

When this return code occurs on an MQSUB call, it can only occur using the MQSO_CREATE option, and in this case the subscription is not created.

Completion Code

MQCC_FAILED

Programmer Response

If this occurs on an MQSUB call, re-issue the MQSUB call using the option MQSO_NEW_PUBLICATIONS_ONLY, which will mean no previously retained publications are sent to this subscription, or fix the SYSTEM.RETAINED.PUB.QUEUE so that messages can be retrieved from it and re-issue the MQSUB call.

If this occurs on an MQSUBRQ call, fix the SYSTEM.RETAINED.PUB.QUEUE so that messages can be retrieved from it and re-issue the MQSUBRQ call.

2526 (09DE) (RC2526): MQRC_RETAINED_NOT_DELIVERED:

Explanation

An MQSUB call which did not use the MQSO_NEW_PUBLICATIONS_ONLY option or an MQSUBRQ call, failed because the retained publications which exist for the topic string subscribed to cannot be delivered to the subscription destination queue and have subsequently failed to be delivered to the dead-letter queue.

When this return code occurs on an MQSUB call, it can only occur using the MQSO_CREATE option, and in this case the subscription is not created.

Completion Code

MQCC_FAILED

Programmer response

Fix the problems with the destination queue and the dead-letter queue and re-issue the MQSUB or MQSUBRQ call.

2527 (09DF) (RC2527): MQRC_RFH_RESTRICTED_FORMAT_ERR:

Explanation

A message was put to a queue containing an MQRFH2 header which included a folder with a restricted format. However, the folder was not in the required format. These restrictions are:
- If NameValueCCSID of the folder is 1208 then only single byte UTF-8 characters are allowed in the folder, group or element names.
- Groups are not allowed in the folder.
- The values of properties may not contain any characters that require escaping.
- Only Unicode character U+0020 will be treated as white space within the folder.
- The folder tag does not contain the content attribute.
- The folder must not contain a property with a null value.

The <mq> folder requires formatting of this restricted form.

Completion Code

MQCC_FAILED

Programmer Response

Change the message to include valid MQRFH2 folders.

2528 (09E0) (RC2528): MQRC_CONNECTION_STOPPED:

Explanation

An MQCTL call was issued to start the asynchronous consumption of messages, but before the connection was ready to consume messages it was stopped by one of the message consumers.

Completion Code

MQCC_FAILED

Programmer Response

If this is an expected condition, no corrective action required. If this is an unexpected condition check whether an MQCTL with Operation MQOP_STOP was issued during the MQCBCT_START callback function.

2529 (09E1) (RC2529): MQRC_ASYNC_UOW_CONFLICT:

Explanation

An MQCTL call with Operation MQOP_START was issued to start the asynchronous consumption of messages, but the connection handle used already has a global unit of work outstanding. MQCTL cannot be used to start asynchronous consumption of messages while a unit of work is in existence unless the MQOP_START_WAIT Operation is used

Completion Code

MQCC_FAILED

Programmer Response

Issue an MQCMIT on the connection handle to commit the unit of work and then reissue the MQCTL call, or issue an MQCTL call using Operation MQOP_START_WAIT to use the unit of work from within the asynchronous consumption callback functions.

2530 (09E2) (RC2530): MQRC_ASYNC_XA_CONFLICT:

Explanation

An MQCTL call with Operation MQOP_START was issued to start the asynchronous consumption of messages, but an external XA syncpoint coordinator has already issued an xa_open call for this connection handle. XA transactions must be done using the MQOP_START_WAIT Operation.

Completion Code

MQCC_FAILED

Programmer Response

Reissue the MQCTL call using Operation MQOP_START_WAIT.

2531 (09E3) (RC2531): MQRC_PUBSUB_INHIBITED:

Explanation

MQSUB, MQOPEN, MQPUT, and MQPUT1 calls are currently inhibited for all publish/subscribe topics, either with the queue manager attribute PSMODE or because processing of publish/subscribe state at queue manager start-up has failed, or has not yet completed.

Completion Code

MQCC_FAILED

Programmer Response

If this queue manager does not intentionally inhibit publish/subscribe, investigate any error messages that describe the failure at queue manager start-up, or wait until start-up processing completes. If the queue manager is a member of cluster, then start-up is not complete until the channel initiator has also started. On z/OS, if you get this return code from the Chinit for the SYSTEM.BROKER.DEFAULT.STREAM queue or topic, then the Chinit is busy processing work, and the pubsub task starts later. Use the DISPLAY PUBSUB command to check the status of the publish/subscribe engine to ensure that it is ready for use. Additionally, on z/OS you might receive an information message CSQM076I.

2532 (09E4) (RC2532): MQRC_MSG_HANDLE_COPY_FAILURE:

Explanation

An MQGET call was issued specifying a valid `MsgHandle` in which to retrieve any properties of the message. After the message had been removed from the queue the application could not allocate enough storage for the properties of the message. The message data is available to the application but the properties are not. Check the queue manager error logs for more information about how much storage was required.

Completion Code

MQCC_WARNING

Programmer response

Raise the memory limit of the application to allow it store the properties.

2533 (09E5) (RC2533): MQRC_DEST_CLASS_NOT_ALTERABLE:
Explanation

An MQSUB call using option MQSO_ALTER was made changing the use of the MQSO_MANAGED option on the subscription. The destination class of a subscription cannot be changed. When the MQSO_MANAGED option is not used, the queue provided can be changed, but the class of destination (managed or not) cannot be changed.

Completion Code

MQCC_FAILED

Programmer Response

Remove the subscription using MQCLOSE and re-create it with MQSUB with the attributes set correctly, or change the use of the MQSO_MANAGED option used on the MQSUB call so that it matches the existing subscription.

2534 (09E6) (RC2534): MQRC_OPERATION_NOT_ALLOWED:

Explanation

An MQCTL call was made with an Operation that is not allowed because of the state of asynchronous consumption on the hConn is currently in.

If Operation was MQOP_RESUME, the operation is not allowed because the state of asynchronous consumption on the hConn is STOPPED. Re-issue MQCTL with the MQOP_START Operation.

If Operation was MQOP_SUSPEND, the operation is not allowed because the state of asynchronous consumption on the hConn is STOPPED. If you need to get your hConn into a SUSPENDED state, issue MQCTL with the MQOP_START Operation followed by MQCTL with MQOP_SUSPEND.

If Operation was MQOP_START, the operation is not allowed because the state of asynchronous consumption on the hConn is SUSPENDED. Re-issue MQCTL with the MQOP_RESUME Operation.

If Operation was MQOP_START_WAIT, the operation is not allowed because either:

- The state of asynchronous consumption on the hConn is SUSPENDED. Re-issue MQCTL with the MQOP_RESUME Operation.
- The state of asynchronous consumption on the hConn is already STARTED. Do not mix the use of MQOP_START and MQOP_START_WAIT within one application.

Completion Code

MQCC_FAILED

Programmer Response

Re-issue the MQCTL call with the correct Operation.

2535 (09E7): MQRC_ACTION_ERROR:
Explanation

An MQPUT call was issued, but the value of the Action field in the PutMsgOpts parameter is not a valid MQACTP_* value.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid value for the field.

2537 (09E9) (RC2537): MQRC_CHANNEL_NOT_AVAILABLE:

Explanation

An MQCONN call was issued from a client to connect to a queue manager but the channel is not currently available. Common causes of this reason code are:
- The channel is currently in stopped state.
- The channel has been stopped by a channel exit.
- The queue manager has reached its maximum allowable limit for this channel from this client.
- The queue manager has reached its maximum allowable limit for this channel.
- The queue manager has reached its maximum allowable limit for all channels.

Completion Code

MQCC_FAILED

Programmer Response

Examine the queue manager and client error logs for messages explaining the cause of the problem.

This reason code is also used to identify the corresponding event message Channel Not Available.

2538 (09EA) (RC2538): MQRC_HOST_NOT_AVAILABLE:

Explanation

An MQCONN call was issued from a client to connect to a queue manager but the attempt to allocate a conversation to the remote system failed. Common causes of this reason code are:

- The listener has not been started on the remote system.
- The connection name in the client channel definition is incorrect.
- The network is currently unavailable.
- A firewall blocking the port, or protocol-specific traffic.
- The security call initializing the IBM MQ client is blocked by a security exit on the SVRCONN channel at the server.

Completion Code

MQCC_FAILED

Programmer Response

Examine the client error log for messages explaining the cause of the problem.

If you are using a Linux server, and receiving a 2538 return code when trying to connect to a queue manager, ensure that you check your internal firewall configuration.

To diagnose the problem, issue the following commands to temporarily turn off the internal Linux firewall :

```
/etc/init.d/iptables save
/etc/init.d/iptables stop
```

To turn the internal Linux firewall back on, issue the command:

```
/etc/init.d/iptables start
```

To permanently turn off the internal Linux firewall, issue the command:

```
chkconfig iptables off
```

2539 (09EB) (RC2539): MQRC_CHANNEL_CONFIG_ERROR:

Explanation

An MQCONN call was issued from a client to connect to a queue manager but the attempt to establish communication failed. Common causes of this reason code are:

- The server and client cannot agree on the channel attributes to use.
- There are errors in one or both of the QM.INI or MQCLIENT.INI configuration files.
- The server machine does not support the code page used by the client.

Completion Code

MQCC_FAILED

Programmer Response

Examine the queue manager and client error logs for messages explaining the cause of the problem.

2540 (09EC) (RC2540): MQRC_UNKNOWN_CHANNEL_NAME:

Explanation

An MQCONN call was issued from a client to connect to a queue manager but the attempt to establish communication failed because the queue manager did not recognize the channel name.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the client is configured to use the correct channel name.

2541 (09ED) (RC2541): MQRC_LOOPING_PUBLICATION:

Explanation

A Distributed Pub/Sub topology has been configured with a combination of Pub/Sub clusters and Pub/Sub Hierarchies such that some, or all, of the queue managers have been connected in a loop. A looping publication has been detected and put onto the dead-letter queue.

Completion Code

MQCC_FAILED

Programmer response

Examine the hierarchy and correct the loop.

2543 (09EF) (RC2543): MQRC_STANDBY_Q_MGR:

Explanation

The application attempted to connect to a standby queue manager instance.

Standby queue manager instances do not accept connections. To connect to the queue manager, you must connect to its active instance.

Completion Code

MQCC_FAILED

Programmer response

Connect the application to an active queue manager instance.

2544 (09F0) (RC2544): MQRC_RECONNECTING:

Explanation

The connection has started reconnecting.

If an event handler has been registered with a reconnecting connection, it is called with this reason code when reconnection attempts begin.

Completion Code

MQCC_WARNING

Programmer response

Let IBM MQ continue with its next reconnection attempt, change the interval before the reconnection, or stop the reconnection. Change any application state that depends on the reconnection.

Note: Reconnection might start while the application is in the middle of an MQI call.

2545 (09F1) (RC2545): MQRC_RECONNECTED:

Explanation

The connection reconnected successfully and all handles are reinstated.

If reconnection succeeds, an event handler registered with the connection is called with this reason code.

Completion Code

MQCC_OK

Programmer response

Set any application state that depends on the reconnection.

Note: Reconnection might finish while the application is in the middle of an MQI call.

2546 (09F2) (RC2546): MQRC_RECONNECT_QMID_MISMATCH:

Explanation

A reconnectable connection specified MQCNO_RECONNECT_Q_MGR and the connection attempted to reconnect to a different queue manager.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the configuration for a reconnectable client resolves to a single queue manager.

If the application does not require reconnection to exactly the same queue manager, use the MQCONNX option MQCNO_RECONNECT.

2547 (09F3) (RC2547): MQRC_RECONNECT_INCOMPATIBLE:

Explanation

An MQI option is incompatible with reconnectable connections.

This error indicates that the option relies on information in a queue manager that is lost during reconnection. For example, the option MQPMO_LOGICAL_ORDER, requires the queue manager to remember information about logical message ordering that is lost during reconnection.

Completion Code

MQCC_FAILED

Programmer response

Modify your application to remove the incompatible option, or do not allow the application to be reconnectable.

2548 (09F4) (RC2548): MQRC_RECONNECT_FAILED:

Explanation

After reconnecting, an error occurred while reinstating the handles for a reconnectable connection.

For example, an attempt to reopen a queue that had been open when the connection broke, failed.

Completion Code

MQCC_FAILED

Programmer response

Investigate the cause of the error in the error logs. Consider using the MQSTAT API to find further details of the failure.

2549 (09F5) (RC2549): MQRC_CALL_INTERRUPTED:

Explanation

MQPUT, MQPUT1, or MQCMIT was interrupted and reconnection processing cannot reestablish a definite outcome.

This reason code is returned to a client that is using a reconnectable connection if the connection is broken between sending the request to the queue manager and receiving the response, and if the outcome is not certain. For example, an interrupted MQPUT of a persistent message outside sync point might or might not have stored the message. Alternatively an interrupted MQPUT1 of a persistent message or message with default persistence (which could be persistent) outside sync point might or might not have stored the message. The timing of the failure affects whether the message remains on the queue or not. If MQCMIT was interrupted the transaction might or might not have been committed.

Completion Code

MQCC_FAILED

Programmer response

Repeat the call following reconnection, but be aware that in some cases, repeating the call might be misleading.

The application design determines the appropriate recovery action. In many cases, getting and putting persistent messages inside sync point resolves indeterminate outcomes. Where persistent messages need to be processed outside sync point, it might be necessary to establish whether the interrupted operation succeeded before the interruption and repeating it if it did not.

2550 (09F6) (RC2550): MQRC_NO_SUBS_MATCHED:

Explanation

An MQPUT or MQPUT1 call was successful but no subscriptions matched the topic.

Completion Code

MQCC_WARNING

Programmer response

No response is required, unless this reason code was not expected by the application that put the message.

2551 (09F7) (RC2551): MQRC_SELECTION_NOT_AVAILABLE:

Explanation

An MQSUB call subscribed to publications using a SelectionString. IBM MQ is unable to accept the call because it does not follow the rules for specifying selection strings, which are documented in Message selector syntax. It is possible that the selection string is acceptable to an extended message selection provider, however no extended message selection provider was available to validate the selection string. If a subscription is being created, the MQSUB fails; otherwise MQSUB completes with a warning.

An MQPUT or MQPUT1 call published a message and at least one subscriber had a content filter but IBM MQ could not determine whether the publication should be delivered to the subscriber (for example, because no extended message selection provider was available to validate the selection string). The MQPUT or MQPUT1 call will fail with MQRC_SELECTION_NOT_AVAILABLE and no subscribers will receive the publication.

Completion Code

MQCC_WARNING or MQCC_FAILED

Programmer response

If it was intended that the selection string should be handled by the extended message selection provider, ensure that the extended message selection provider is correctly configured and running. If extended message selection was not intended, see Message selector syntax and ensure that you have correctly followed the rules for specifying selection strings.

If a subscription is being resumed, the subscription will not be delivered any messages until a extended message selection provider is available and a message matches the SelectionString of the resumed subscription.

2552 (09F8) (RC2552): MQRC_CHANNEL_SSL_WARNING:
Explanation

An SSL security event has occurred. This is not fatal to an SSL connection but is likely to be of interest to an administrator.

Completion code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel SSL Warning.

2553 (09F9) (RC2553): MQRC_OCSP_URL_ERROR:
Explanation

The OCSPResponderURL field does not contain a correctly formatted HTTP URL.

Completion code

MQCC_FAILED

Programmer response

Check and correct the OCSPResponderURL. If you do not intend to access an OCSP responder, set the AuthInfoType of the authentication information object to MQAIT_CRL_LDAP.

2554 (09FA) (RC2554): MQRC_CONTENT_ERROR:

Explanation

There are 2 explanations for reason code 2554:
1. An MQPUT call was issued with a message where the content could not be parsed to determine whether the message should be delivered to a subscriber with an extended message selector. No subscribers will receive the publication.
2. MQRC_CONTENT_ERROR can be returned from MQSUB and MQSUBRQ if a selection string selecting on the content of the message was specified.

Completion Code

MQCC_FAILED

Programmer response

There are 2 programmer responses for reason code 2554 because there are two causes:
1. If reason code 2554 was issued because of reason 1 then check for error messages from the extended message selection provider and ensure that the message content is well formed before retrying the operation.
2. If reason code 2554 was issued because of reason 2 then because the error occurred at the time that the retained message was published, either a system administrator must clear the retained queue, or you cannot specify a selection string selecting on the content.

2555 (09FB) (RC2555): MQRC_RECONNECT_Q_MGR_REQD:

Explanation

The `MQCNO_RECONNECT_Q_MGR` option is required.

An option, such `MQMO_MATCH_MSG_TOKEN` in an MQGET call or opening a durable subscription, was specified in the client program that requires re-connection to the same queue manager.

Completion Code

`MQCC_FAILED`

Programmer response

Change the MQCONNX call to use `MQCNO_RECONNECT_Q_MGR`, or modify the client program not to use the conflicting option.

2556 (09FC) (RC2556): MQRC_RECONNECT_TIMED_OUT:

Explanation

A reconnection attempt timed out.

The failure might occur in any MQI verb if a connection is configured to reconnect. You can customize the timeout in the MQClient.ini file

Completion Code

MQCC_FAILED

Programmer response

Look at the error logs to find out why reconnection did not complete within the time limit.

2557 (09FD) (RC2557): MQRC_PUBLISH_EXIT_ERROR:

Explanation

A publish exit function returned an invalid response code, or failed in some other way. This can be returned from the MQPUT, MQPUT1, MQSUB and MQSUBRQ function calls. This reason code does not occur on IBM MQ for z/OS.

Completion Code

MQCC_FAILED

Programmer response

Check the publish exit logic to ensure that the exit is returning valid values in the ExitResponse field of the MQPSXP structure. Consult the IBM MQ error log files and FFST records for more details about the problem.

2558 (09FE) (RC2558): MQRC_COMMINFO_ERROR:

Explanation

The configuration of either the name of the COMMINFO object or the object itself is incorrect.

Completion Code

MQCC_FAILED

Programmer response

Check the configuration of the TOPIC and COMMINFO objects and retry the operation.

2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY:

Explanation

An attempt was made to use a topic which is defined as multicast only in a non-multicast way. Possible causes for this error are:

1. An MQPUT1 call was issued to the topic
2. An MQOPEN call was issued using the MQOO_NO_MULTICAST option
3. An MQSUB call was issued using the MQSO_NO_MULTICAST option
4. The application is connected directly through bindings, that is, there is no client connection
5. The application is being run from a release prior to Version 7.1

Completion Code

MQCC_FAILED

Programmer response

Either change the topic definition to enable non-multicast, or change the application.

2561 (0A01) (RC2561): MQRC_DATA_SET_NOT_AVAILABLE:

Explanation

An IBM MQI call or command was issued to operate on a shared queue, but the call failed because the data for the shared message has been offloaded to a shared message data set that is temporarily unavailable to the current queue manager. This can occur either because of a problem in accessing the data set or because the data set was previously found to be damaged, and is awaiting completion of recovery processing.

This return code can also occur if the shared message data set has not been defined for the queue manager being used. You might be using the wrong queue manager in the queue-sharing group.

- This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

The problem is temporary; wait a short while, and then retry the operation.

Use `DIS CFSTRUCT(...) SMDSCONN(*)` to display the status of the SMDS connection.

To start the connection if the STATUS is not OPEN, use `STA SMDSCONN(*) CFSTRUCT(...)`.

Use `DISPLAY CFSTATUS(...) TYPE(SMDS)` and check the status is active on the queue manager that you are using.

2562 (0A02) (RC2562): MQRC_GROUPING_NOT_ALLOWED:

Explanation

An MQPUT call was issued to put a grouped message to a handle which is publishing over multicast.

Completion Code

MQCC_FAILED

Programmer response

Either change the topic definition to disable multicast or change the application to not use grouped messages.

2563 (0A03) (RC2563): MQRC_GROUP_ADDRESS_ERROR:

Explanation

An MQOPEN or MQSUB call was issued to a multicast topic which has been defined with an incorrect group address field.

Completion Code

MQCC_FAILED

Programmer response

Correct the group address field in the COMMINFO definition linked to the TOPIC object.

2564 (0A04) (RC2564): MQRC_MULTICAST_CONFIG_ERROR:

Explanation

An MQOPEN, MQSUB or MQPUT call was issued which invoked the multicast component. The call failed because the multicast configuration is incorrect.

Completion Code

MQCC_FAILED

Programmer response

Check the multicast configuration and error logs and retry the operation.

2565 (0A05) (RC2565): MQRC_MULTICAST_INTERFACE_ERROR:

Explanation

An MQOPEN, MQSUB or MQPUT call was made which attempted to a network interface for multicast. The interface returned an error. Possible causes for the error are:

1. The required network interface does not exist.
2. The interface is not active.
3. The interface does not support the required IP version.

Completion Code

MQCC_FAILED

Programmer response

Verify that the IP address and the system network configuration are valid. Check the multicast configuration and error logs and retry the operation.

2566 (0A06) (RC2566): MQRC_MULTICAST_SEND_ERROR:

Explanation

An MQPUT call was made which attempted to send multicast traffic over the network. The system failed to send one or more network packets.

Completion Code

MQCC_FAILED

Programmer response

Verify that the IP address and the system network configuration are valid. Check the multicast configuration and error logs and retry the operation.

2567 (0A07) (RC2567): MQRC_MULTICAST_INTERNAL_ERROR:

Explanation

An MQOPEN, MQSUB or MQPUT call was issued which invoked the multicast component. An internal error occurred which prevented the operation completing successfully.

Completion Code

MQCC_FAILED

Programmer response

Tell the systems administrator.

2568 (0A08) (RC2568): MQRC_CONNECTION_NOT_AVAILABLE:

Explanation

An MQCONN or MQCONNX call was made when the queue manager was unable to provide a connection of the requested connection type on the current installation. A client connection cannot be made on a server only installation. A local connection cannot be made on a client only installation.

This error can also occur when IBM MQ fails an attempt to load a library from the installation that the requested queue manager is associated with.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the connection type requested is applicable to the type of installation. If the connection type is applicable to the installation then consult the error log for more information about the nature of the error.

2569 (0A09) (RC2569): MQRC_SYNCPOINT_NOT_ALLOWED:

Explanation

An MQPUT or MQPUT1 call using MQPMO_SYNCPOINT was made to a topic that is defined as MCAST(ENABLED). This is not allowed.

Completion Code

MQCC_FAILED

Programmer response

Change the application to use MQPMO_NO_SYNCPOINT, or alter the topic to disable the use of Multicast and retry the operation.

2577 (0A11) (RC2577): MQRC_CHANNEL_BLOCKED:

Explanation

An inbound channel attempted to connect to the queue manager but was blocked due to matching a Channel Authentication rule.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Blocked.

2578 (0A12) (RC2578): MQRC_CHANNEL_BLOCKED_WARNING:

Explanation

An inbound channel attempted to connect to the queue manager and would have been blocked due to matching a Channel Authentication rule, however the rule was defined with WARN(YES) so the rule did not block the connection.

Completion Code

MQCC_WARNING

Programmer response

None. This reason code is only used to identify the corresponding event message Channel Blocked.

2583 (0A17) (RC2583): MQRC_INSTALLATION_MISMATCH:

Explanation

The application attempted to connect to a queue manager that is not associated with the same IBM MQ installation as the loaded libraries.

Completion Code

MQCC_FAILED

Programmer response

An application must use the libraries from the installation the queue manager is associated with. If the *AMQ_SINGLE_INSTALLATION* environment variable is set, you must ensure that the application connects only to queue managers associated with a single installation. Otherwise, if IBM MQ is unable to automatically locate the correct libraries, you must modify the application, or the library search path, to ensure that the correct libraries are used.

2587 (0A1B) (RC2587): MQRC_HMSG_NOT_AVAILABLE:

Explanation

On an MQGET, MQPUT, or MQPUT1 call, a message handle supplied is not valid with the installation the queue manager is associated with. The message handle was created by MQCRTMH specifying the MQHC_UNASSOCIATED_HCONN option. It can be used only with queue managers associated with the first installation used in the process.

Completion Code

MQCC_FAILED

Programmer response

To pass properties between two queue managers associated with different installations, convert the message handle retrieved using MQGET into a buffer using the MQMHBUF call. Then pass that buffer into the MQPUT or MQPUT1 call of the other queue manager. Alternatively, use the **`setmqm`** command to associate one of the queue managers with the installation that the other queue manager is using. Using the **`setmqm`** command might change the version of IBM MQ that the queue manager uses.

2589 (0A1D) (RC2589) MQRC_INSTALLATION_MISSING:

Explanation

On an MQCONN or MQCONNX call, an attempt was made to connect to a queue manager where the associated installation is no longer installed.

Completion Code

MQCC_FAILED

Programmer response

Associate the queue manager with a different installation using the **setmqm** command before attempting to connect to the queue manager again.

2590 (0A1E) (RC2590): MQRC_FASTPATH_NOT_AVAILABLE:

Explanation

On an MQCONNX call, the MQCNO_FASTPATH_BINDING option was specified. However, a fastpath connection to the queue manager cannot be made. This issue can occur when a non-fastpath connection to a queue manager was made in the process before this MQCONNX call.

Completion Code

MQCC_FAILED

Programmer response

Either change all MQCONNX calls within the process to be fastpath, or use the *AMQ_SINGLE_INSTALLATION* environment variable to restrict connections to a single installation, allowing the queue manager to accept fastpath and non-fastpath connections from the same process, in any order.

2591 (0A1F) (RC2591): MQRC_CIPHER_SPEC_NOT_SUITE_B:

Explanation

A client application is configured for NSA Suite B compliant operation but the CipherSpec for the client connection channel is not permitted at the configured Suite B security level. This can occur for Suite B CipherSpecs which fall outside the currently configured security level, for example if ECDHE_ECDSA_AES_128_GCM_SHA256 (which is 128-bit Suite B) is used when only the 192-bit Suite B security level is configured

For more information about which CipherSpecs are Suite B compliant, refer to Specifying CipherSpecs.

Completion Code

MQCC_FAILED

Programmer response

Select an appropriate CipherSpec which is permitted at the configured Suite B security level.

2592 (0A20) (RC2592): MQRC_SUITE_B_ERROR:

Explanation

The configuration of Suite B is invalid. For example, an unrecognized value was specified in the **MQSUITEB** environment variable, the **EncryptionPolicySuiteB** SSL stanza setting or the MQSCO **EncryptionPolicySuiteB** field.

Completion Code

MQCC_FAILED

Programmer response

Determine the fault in the Suite B configuration and amend.

2593 (0A21)(RC2593): MQRC_CERT_VAL_POLICY_ERROR:

Explanation

The certificate validation policy configuration is invalid. An unrecognized or unsupported value was specified in the **MQCERTVPOL** environment variable, the **CertificateValPolicy** SSL stanza setting or the MQSCO **CertificateValPolicy** field.

Completion Code

MQCC_FAILED

Programmer response

Specify a valid certificate validation policy which is supported on the current platform.

2594 (0A22)(RC2594): MQRC_PASSWORD_PROTECTION_ERROR:

Explanation

An MQCONN or MQCONNX call was issued from a client connected application, but it failed to agree a password protection algorithm with the queue manager. For unencrypted channels, IBM MQ Version 8.0 clients try to agree a password protection mechanism to avoid sending passwords in plain text across a network.

The usual cause of this error is that the user has set the PasswordProtection attribute in the Channels stanza of mqclient.ini (or qm.ini) to ALWAYS, but the version of IBM MQ that is installed on the remote system does not support password protection.

Java and JMS clients must enable MQCSP authentication mode in order to use the PasswordProtection feature. See Connection authentication with the Java client.

Completion Code

MQCC_FAILED

Programmer response

Consider changing the PasswordProtection attribute or use SSL/TLS to protect passwords instead. If you are using SSL/TLS, you must not use a null cipher because it would send passwords in plain text which provides no protection.

More information can be found in the error log in message AMQ9296.

2595 (0A23)(RC2595): MQRC_CSP_ERROR:

Explanation

The connect call failed because the MQCSP structure was not valid for one of the following reasons:
- The **`StrucId`** field is not MQCSP_STRUC_ID
- The **`Version`** field specifies a value that is not valid or not supported.
- The **`AuthenticationType`** field specifies a value that is not valid or not supported.
- The user identifier is incorrectly specified.
- The password is incorrectly specified.

Completion Code

MQCC_FAILED

Programmer response

Ensure that the MQCSP structure is correct.

z/OS On z/OS:
- Check that the IBM MQ libraries in STEPLIB are at the same or a higher level than the queue manager.
- If you are using USS, check that the LIBPATH has matching libraries, for example `LIBPATH=$LIBPATH:"/mqm/V8R0M0/java/lib/"`.

2596 (0A24)(RC2596): MQRC_CERT_LABEL_NOT_ALLOWED:

Explanation

The channel definition specifies a certificate label but the environment does not support certificate label configuration.

Completion Code

MQCC_FAILED

Programmer response

Either, remove the certificate label from the channel definition, or change the configuration to ignore the label.

2598 (0A26)(RC2598): MQRC_ADMIN_TOPIC_STRING_ERROR: V 8.0.0.2

Explanation

This error can occur when calling MQSUB or MQOPEN. Publishing to an IBM MQ administrative topic string, starting `$SYS/MQ/` is not permitted.

When subscribing to an IBM MQ administrative topic string, the use of wildcard characters is restricted. See System topics for monitoring and activity trace for details.

Completion Code

MQCC_FAILED

Programmer response

Change the configuration to publish to an administrative topic string that does not start `$SYS/MQ/`.

6100 (17D4) (RC6100): MQRC_REOPEN_EXCL_INPUT_ERROR:

Explanation

An open object does not have the correct ImqObject **open options** and requires one or more additional options. An implicit reopen is required but closure has been prevented.

Closure has been prevented because the queue is open for exclusive input and closure might result in the queue being accessed by another process or thread, before the queue is reopened by the process or thread that presently has access.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Set the **open options** explicitly to cover all eventualities so that implicit reopening is not required.

6101 (17D5) (RC6101): MQRC_REOPEN_INQUIRE_ERROR:

Explanation

An open object does not have the correct ImqObject **open options** and requires one or more additional options. An implicit reopen is required but closure has been prevented.

Closure has been prevented because one or more characteristics of the object need to be checked dynamically prior to closure, and the **open options** do not already include MQOO_INQUIRE.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Set the **open options** explicitly to include MQOO_INQUIRE.

6102 (17D6) (RC6102): MQRC_REOPEN_SAVED_CONTEXT_ERR:

Explanation

An open object does not have the correct ImqObject **open options** and requires one or more additional options. An implicit reopen is required but closure has been prevented.

Closure has been prevented because the queue is open with MQOO_SAVE_ALL_CONTEXT, and a destructive get has been performed previously. This has caused retained state information to be associated with the open queue and this information would be destroyed by closure.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Set the **open options** explicitly to cover all eventualities so that implicit reopening is not required.

6103 (17D7) (RC6103): MQRC_REOPEN_TEMPORARY_Q_ERROR:

Explanation

An open object does not have the correct ImqObject **open options** and requires one or more additional options. An implicit reopen is required but closure has been prevented.

Closure has been prevented because the queue is a local queue of the definition type MQQDT_TEMPORARY_DYNAMIC, that would be destroyed by closure.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Set the **open options** explicitly to cover all eventualities so that implicit reopening is not required.

6104 (17D8) (RC6104): MQRC_ATTRIBUTE_LOCKED:

Explanation

An attempt has been made to change the value of an attribute of an object while that object is open, or, for an ImqQueueManager object, while that object is connected. Certain attributes cannot be changed in these circumstances. Close or disconnect the object (as appropriate) before changing the attribute value.

An object might have been connected, opened, or both unexpectedly and implicitly to perform an MQINQ call. Check the attribute cross-reference table in C++ and MQI cross-reference to determine whether any of your method invocations result in an MQINQ call.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Include MQOO_INQUIRE in the ImqObject **open options** and set them earlier.

6105 (17D9) (RC6105): MQRC_CURSOR_NOT_VALID:

Explanation

The browse cursor for an open queue has been invalidated since it was last used by an implicit reopen.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Set the ImqObject **open options** explicitly to cover all eventualities so that implicit reopening is not required.

6106 (17DA) (RC6106): MQRC_ENCODING_ERROR:

Explanation

The encoding of the (next) message item needs to be MQENC_NATIVE for pasting.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6107 (17DB) (RC6107): MQRC_STRUC_ID_ERROR:

Explanation

The structure id for the (next) message item, which is derived from the 4 characters beginning at the data pointer, is either missing or is inconsistent with the class of object into which the item is being pasted.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6108 (17DC) (RC6108): MQRC_NULL_POINTER:

Explanation

A null pointer has been supplied where a nonnull pointer is either required or implied.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6109 (17DD) (RC6109): MQRC_NO_CONNECTION_REFERENCE:

Explanation

The **connection reference** is null. A connection to an ImqQueueManager object is required.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6110 (17DE) (RC6110): MQRC_NO_BUFFER:

Explanation

No buffer is available. For an ImqCache object, one cannot be allocated, denoting an internal inconsistency in the object state that should not occur.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6111 (17DF) (RC6111): MQRC_BINARY_DATA_LENGTH_ERROR:

Explanation

The length of the binary data is inconsistent with the length of the target attribute. Zero is a correct length for all attributes.

- The correct length for an **accounting token** is MQ_ACCOUNTING_TOKEN_LENGTH.
- The correct length for an **alternate security id** is MQ_SECURITY_ID_LENGTH.
- The correct length for a **correlation id** is MQ_CORREL_ID_LENGTH.
- The correct length for a **facility token** is MQ_FACILITY_LENGTH.
- The correct length for a **group id** is MQ_GROUP_ID_LENGTH.
- The correct length for a **message id** is MQ_MSG_ID_LENGTH.
- The correct length for an **instance id** is MQ_OBJECT_INSTANCE_ID_LENGTH.
- The correct length for a **transaction instance id** is MQ_TRAN_INSTANCE_ID_LENGTH.
- The correct length for a **message token** is MQ_MSG_TOKEN_LENGTH.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6112 (17E0) (RC6112): MQRC_BUFFER_NOT_AUTOMATIC:

Explanation

A user-defined (and managed) buffer cannot be resized. A user-defined buffer can only be replaced or withdrawn. A buffer must be automatic (system-managed) before it can be resized.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

6113 (17E1) (RC6113): MQRC_INSUFFICIENT_BUFFER:

Explanation

There is insufficient buffer space available after the data pointer to accommodate the request. This might be because the buffer cannot be resized.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6114 (17E2) (RC6114): MQRC_INSUFFICIENT_DATA:

Explanation

There is insufficient data after the data pointer to accommodate the request.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6115 (17E3) (RC6115): MQRC_DATA_TRUNCATED:

Explanation

Data has been truncated when copying from one buffer to another. This might be because the target buffer cannot be resized, or because there is a problem addressing one or other buffer, or because a buffer is being downsized with a smaller replacement.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6116 (17E4) (RC6116): MQRC_ZERO_LENGTH:

Explanation

A zero length has been supplied where a positive length is either required or implied.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6117 (17E5) (RC6117): MQRC_NEGATIVE_LENGTH:

Explanation

A negative length has been supplied where a zero or positive length is required.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6118 (17E6) (RC6118): MQRC_NEGATIVE_OFFSET:

Explanation

A negative offset has been supplied where a zero or positive offset is required.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6119 (17E7) (RC6119): MQRC_INCONSISTENT_FORMAT:

Explanation

The format of the (next) message item is inconsistent with the class of object into which the item is being pasted.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6120 (17E8) (RC6120): MQRC_INCONSISTENT_OBJECT_STATE:

Explanation

There is an inconsistency between this object, which is open, and the referenced ImqQueueManager object, which is not connected.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6121 (17E9) (RC6121): MQRC_CONTEXT_OBJECT_NOT_VALID:

Explanation

The ImqPutMessageOptions **context reference** does not reference a valid ImqQueue object. The object has been previously destroyed.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6122 (17EA) (RC6122): MQRC_CONTEXT_OPEN_ERROR:

Explanation

The ImqPutMessageOptions **context reference** references an ImqQueue object that could not be opened to establish a context. This might be because the ImqQueue object has inappropriate **open options**. Inspect the referenced object **reason code** to establish the cause.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6123 (17EB) (RC6123): MQRC_STRUC_LENGTH_ERROR:

Explanation

The length of a data structure is inconsistent with its content. For an MQRMH, the length is insufficient to contain the fixed fields and all offset data.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

6124 (17EC) (RC6124): MQRC_NOT_CONNECTED:

Explanation

A method failed because a required connection to a queue manager was not available, and a connection cannot be established implicitly because the IMQ_IMPL_CONN flag of the ImqQueueManager **behavior** class attribute is FALSE.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Establish a connection to a queue manager and retry.

6125 (17ED) (RC6125): MQRC_NOT_OPEN:

Explanation

A method failed because an object was not open, and opening cannot be accomplished implicitly because the IMQ_IMPL_OPEN flag of the ImqObject **behavior** class attribute is FALSE.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Open the object and retry.

6126 (17EE) (RC6126): MQRC_DISTRIBUTION_LIST_EMPTY:

Explanation

An ImqDistributionList failed to open because there are no ImqQueue objects referenced.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Establish at least one ImqQueue object in which the **distribution list reference** addresses the ImqDistributionList object, and retry.

6127 (17EF) (RC6127): MQRC_INCONSISTENT_OPEN_OPTIONS:

Explanation

A method failed because the object is open, and the ImqObject **open options** are inconsistent with the required operation. The object cannot be reopened implicitly because the IMQ_IMPL_OPEN flag of the ImqObject **behavior** class attribute is false.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Open the object with appropriate ImqObject **open options** and retry.

6128 (17FO) (RC6128): MQRC_WRONG_VERSION:

Explanation

A method failed because a version number specified or encountered is either incorrect or not supported.

For the ImqCICSBridgeHeader class, the problem is with the **version** attribute.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

If you are specifying a version number, use one that is supported by the class. If you are receiving message data from another program, ensure that both programs are using consistent and supported version numbers.

6129 (17F1) (RC6129): MQRC_REFERENCE_ERROR:

Explanation

An object reference is invalid.

There is a problem with the address of a referenced object. At the time of use, the address of the object is nonnull, but is invalid and cannot be used for its intended purpose.

This reason code occurs in the IBM MQ C++ environment.

Completion Code

MQCC_FAILED

Programmer response

Check that the referenced object is neither deleted nor out of scope, or remove the reference by supplying a null address value.

PCF reason codes

Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

For more information about PCFs, see Introduction to Programmable Command Formats, Automating administration tasks, and Using Programmable Command Formats.

The following is a list of PCF reason codes, in numeric order, providing detailed information to help you understand them, including:
- An explanation of the circumstances that have caused the code to be raised
- The associated completion code
- Suggested programmer actions in response to the code

Related reference:

"API completion and reason codes" on page 1314
For each call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

"Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes" on page 1622
IBM MQ can use Secure Sockets Layer (SSL) with the various communication protocols. Use this topic to identify the error codes that can be returned by SSL.

"WCF custom channel exceptions" on page 1627
Diagnostic messages are listed in this topic in numeric order, grouped according to the part of the WCF custom channel from which they originate.

Related information:

IBM MQ AMQ messages

z/OS IBM MQ for z/OS messages, completion, and reason codes

3001 (0BB9) (RC3001): MQRCCF_CFH_TYPE_ERROR

Explanation

Type not valid.

The MQCFH *Type* field value was not valid.

Programmer response

Specify a valid type.

3002 (0BBA) (RC3002): MQRCCF_CFH_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFH *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3003 (0BBB) (RC3003): MQRCCF_CFH_VERSION_ERROR

Explanation

Structure version number is not valid.

The MQCFH *Version* field value was not valid.

Note that z/OS requires MQCFH_VERSION_3.

Programmer response

Specify a valid structure version number.

3004 (0BBC) (RC3004): MQRCCF_CFH_MSG_SEQ_NUMBER_ERR

Explanation

Message sequence number not valid.

The MQCFH *MsgSeqNumber* field value was not valid.

Programmer response

Specify a valid message sequence number.

3005 (0BBD) (RC3005): MQRCCF_CFH_CONTROL_ERROR

Explanation

Control option not valid.

The MQCFH *Control* field value was not valid.

Programmer response

Specify a valid control option.

3006 (0BBE) (RC3006): MQRCCF_CFH_PARM_COUNT_ERROR

Explanation

Parameter count not valid.

The MQCFH *ParameterCount* field value was not valid.

Programmer response

Specify a valid parameter count.

3007 (0BBF) (RC3007): MQRCCF_CFH_COMMAND_ERROR

Explanation

Command identifier not valid.

The MQCFH *Command* field value was not valid.

Programmer response

Specify a valid command identifier.

3008 (0BC0) (RC3008): MQRCCF_COMMAND_FAILED

Explanation

Command failed.

The command has failed.

Programmer response

Refer to the previous error messages for this command.

3009 (0BC1) (RC3009): MQRCCF_CFIN_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFIN or MQCFIN64 *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3010 (0BC2) (RC3010): MQRCCF_CFST_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFST *StrucLength* field value was not valid. The value was not a multiple of four or was inconsistent with the MQCFST *StringLength* field value.

Programmer response

Specify a valid structure length.

3011 (0BC3) (RC3011): MQRCCF_CFST_STRING_LENGTH_ERR

Explanation

String length not valid.

The MQCFST *StringLength* field value was not valid. The value was negative or greater than the maximum permitted length of the parameter specified in the *Parameter* field.

Programmer response

Specify a valid string length for the parameter.

3012 (0BC4) (RC3012): MQRCCF_FORCE_VALUE_ERROR

Explanation

Force value not valid.

The force value specified was not valid.

Programmer response

Specify a valid force value.

3013 (0BC5) (RC3013): MQRCCF_STRUCTURE_TYPE_ERROR

Explanation

Structure type not valid.

The structure *Type* value was not valid.

Programmer response

Specify a valid structure type.

3014 (0BC6) (RC3014): MQRCCF_CFIN_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFIN or MQCFIN64 *Parameter* field value was not valid.

Programmer response

Specify a valid parameter identifier.

3015 (0BC7) (RC3015): MQRCCF_CFST_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFST *Parameter* field value was not valid.

Programmer response

Specify a valid parameter identifier.

3016 (0BC8) (RC3016): MQRCCF_MSG_LENGTH_ERROR

Explanation

Message length not valid.

The message data length was inconsistent with the length implied by the parameters in the message, or a positional parameter was out of sequence.

Programmer response

Specify a valid message length, and check that positional parameters are in the correct sequence.

3017 (0BC9) (RC3017): MQRCCF_CFIN_DUPLICATE_PARM

Explanation

Duplicate parameter.

Two MQCFIN or MQCFIN64 or MQCFIL or MQCFIL64 structures, or any two of those types of structure, with the same parameter identifier were present.

Programmer response

Check for and remove duplicate parameters.

3018 (0BCA) (RC3018): MQRCCF_CFST_DUPLICATE_PARM

Explanation

Duplicate parameter.

Two MQCFST structures, or an MQCFSL followed by an MQCFST structure, with the same parameter identifier were present.

Programmer response

Check for and remove duplicate parameters.

3019 (0BCB) (RC3019): MQRCCF_PARM_COUNT_TOO_SMALL

Explanation

Parameter count too small.

The MQCFH *ParameterCount* field value was less than the minimum required for the command.

Programmer response

Specify a parameter count that is valid for the command.

3020 (0BCC) (RC3020): MQRCCF_PARM_COUNT_TOO_BIG

Explanation

Parameter count too big.

The MQCFH *ParameterCount* field value was more than the maximum for the command.

Programmer response

Specify a parameter count that is valid for the command.

3021 (0BCD) (RC3021): MQRCCF_Q_ALREADY_IN_CELL

Explanation

Queue already exists in cell.

An attempt was made to define a queue with cell scope, or to change the scope of an existing queue from queue-manager scope to cell scope, but a queue with that name already existed in the cell.

Programmer response

Do one of the following:
- Delete the existing queue and retry the operation.
- Change the scope of the existing queue from cell to queue-manager and retry the operation.
- Create the new queue with a different name.

3022 (0BCE) (RC3022): MQRCCF_Q_TYPE_ERROR

Explanation

Queue type not valid.

The *QType* value was not valid.

Programmer response

Specify a valid queue type.

3023 (0BCF) (RC3023): MQRCCF_MD_FORMAT_ERROR

Explanation

Format not valid.

The MQMD *Format* field value was not MQFMT_ADMIN.

Programmer response

Specify the valid format.

3024 (0BD0) (RC3024): MQRCCF_CFSL_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFSL *StrucLength* field value was not valid. The value was not a multiple of four or was inconsistent with the MQCFSL *StringLength* field value.

Programmer response

Specify a valid structure length.

3025 (0BD1) (RC3025): MQRCCF_REPLACE_VALUE_ERROR

Explanation

Replace value not valid.

The *Replace* value was not valid.

Programmer response

Specify a valid replace value.

3026 (0BD2) (RC3026): MQRCCF_CFIL_DUPLICATE_VALUE

Explanation

Duplicate parameter value.

In the MQCFIL or MQCFIL64 structure, there was a duplicate parameter value in the list.

Programmer response

Check for and remove duplicate parameter values.

3027 (0BD3) (RC3027): MQRCCF_CFIL_COUNT_ERROR

Explanation

Count of parameter values not valid.

The MQCFIL or MQCFIL64 *Count* field value was not valid. The value was negative or greater than the maximum permitted for the parameter specified in the *Parameter* field.

Programmer response

Specify a valid count for the parameter.

3028 (0BD4) (RC3028): MQRCCF_CFIL_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFIL or MQCFIL64 *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3029 (0BD5) (RC3029): MQRCCF_MODE_VALUE_ERROR

Explanation

Mode value not valid.

The *Mode* value was not valid.

Programmer response

Specify a valid mode value.

3029 (0BD5) (RC3029): MQRCCF_QUIESCE_VALUE_ERROR

Explanation

Former name for MQRCCF_MODE_VALUE_ERROR.

3030 (0BD6) (RC3030): MQRCCF_MSG_SEQ_NUMBER_ERROR

Explanation

Message sequence number not valid.

The message sequence number parameter value was not valid.

Programmer response

Specify a valid message sequence number.

3031 (0BD7) (RC3031): MQRCCF_PING_DATA_COUNT_ERROR

Explanation

Data count not valid.

The Ping Channel `DataCount` value was not valid.

Programmer response

Specify a valid data count value.

3032 (0BD8) (RC3032): MQRCCF_PING_DATA_COMPARE_ERROR

Explanation

Ping Channel command failed.

The Ping Channel command failed with a data compare error. The data offset that failed is returned in the message (with parameter identifier MQIACF_ERROR_OFFSET).

Programmer response

Consult your systems administrator.

3033 (0BD9) (RC3033): MQRCCF_CFSL_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFSL `Parameter` field value was not valid.

Programmer response

Specify a valid parameter identifier.

3034 (0BDA) (RC3034): MQRCCF_CHANNEL_TYPE_ERROR

Explanation

Channel type not valid.

The `ChannelType` specified was not valid, or did not match the type of an existing channel being copied, changed or replaced, or the command and the specified disposition cannot be used with that type of channel.

Programmer response

Specify a valid channel name, type, or disposition.

3035 (0BDB) (RC3035): MQRCCF_PARM_SEQUENCE_ERROR

Explanation

Parameter sequence not valid.

The sequence of parameters is not valid for this command.

Programmer response

Specify the positional parameters in a valid sequence for the command.

3036 (0BDC) (RC3036): MQRCCF_XMIT_PROTOCOL_TYPE_ERR

Explanation

Transmission protocol type not valid.

The *TransportType* value was not valid.

Programmer response

Specify a valid transmission protocol type.

3037 (0BDD) (RC3037): MQRCCF_BATCH_SIZE_ERROR

Explanation

Batch size not valid.

The batch size specified was not valid.

Programmer response

Specify a valid batch size value.

3038 (0BDE) (RC3038): MQRCCF_DISC_INT_ERROR

Explanation

Disconnection interval not valid.

The disconnection interval specified was not valid.

Programmer response

Specify a valid disconnection interval.

3039 (0BDF) (RC3039): MQRCCF_SHORT_RETRY_ERROR

Explanation

Short retry count not valid.

The *ShortRetryCount* value was not valid.

Programmer response

Specify a valid short retry count value.

3040 (0BE0) (RC3040): MQRCCF_SHORT_TIMER_ERROR

Explanation

Short timer value not valid.

The *ShortRetryInterval* value was not valid.

Programmer response

Specify a valid short timer value.

3041 (0BE1) (RC3041): MQRCCF_LONG_RETRY_ERROR

Explanation

Long retry count not valid.

The long retry count value specified was not valid.

Programmer response

Specify a valid long retry count value.

3042 (0BE2) (RC3042): MQRCCF_LONG_TIMER_ERROR

Explanation

Long timer not valid.

The long timer (long retry wait interval) value specified was not valid.

Programmer response

Specify a valid long timer value.

3043 (0BE3) (RC3043): MQRCCF_SEQ_NUMBER_WRAP_ERROR

Explanation

Sequence wrap number not valid.

The *SeqNumberWrap* value was not valid.

Programmer response

Specify a valid sequence wrap number.

3044 (0BE4) (RC3044): MQRCCF_MAX_MSG_LENGTH_ERROR

Explanation

Maximum message length not valid.

The maximum message length value specified was not valid.

Programmer response

Specify a valid maximum message length.

3045 (0BE5) (RC3045): MQRCCF_PUT_AUTH_ERROR

Explanation

Put authority value not valid.

The *PutAuthority* value was not valid.

Programmer response

Specify a valid authority value.

3046 (0BE6) (RC3046): MQRCCF_PURGE_VALUE_ERROR

Explanation

Purge value not valid.

The *Purge* value was not valid.

Programmer response

Specify a valid purge value.

3047 (0BE7) (RC3047): MQRCCF_CFIL_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFIL or MQCFIL64 `Parameter` field value was not valid, or specifies a parameter that cannot be filtered, or that is also specified as a parameter to select a subset of objects.

Programmer response

Specify a valid parameter identifier.

3048 (0BE8) (RC3048): MQRCCF_MSG_TRUNCATED

Explanation

Message truncated.

The command server received a message that is larger than its maximum valid message size.

Programmer response

Check the message contents are correct.

3049 (0BE9) (RC3049): MQRCCF_CCSID_ERROR

Explanation

Coded character-set identifier error.

In a command message, one of the following occurred:
- The *CodedCharSetId* field in the message descriptor of the command does not match the coded character-set identifier of the queue manager at which the command is being processed, or
- The *CodedCharSetId* field in a string parameter structure within the message text of the command is not
  - MQCCSI_DEFAULT, or
  - the coded character-set identifier of the queue manager at which the command is being processed, as in the *CodedCharSetId* field in the message descriptor.

The error response message contains the correct value.

This reason can also occur if a ping cannot be performed because the coded character-set identifiers are not compatible. In this case the correct value is not returned.

Programmer response

Construct the command with the correct coded character-set identifier, and specify this in the message descriptor when sending the command. For ping, use a suitable coded character-set identifier.

3050 (0BEA) (RC3050): MQRCCF_ENCODING_ERROR

Explanation

Encoding error.

The *Encoding* field in the message descriptor of the command does not match that required for the platform at which the command is being processed.

Programmer response

Construct the command with the correct encoding, and specify this in the message descriptor when sending the command.

3052 (0BEC) (RC3052): MQRCCF_DATA_CONV_VALUE_ERROR

Explanation

Data conversion value not valid.

The value specified for *DataConversion* is not valid.

Programmer response

Specify a valid value.

3053 (0BED) (RC3053): MQRCCF_INDOUBT_VALUE_ERROR

Explanation

In-doubt value not valid.

The value specified for *InDoubt* is not valid.

Programmer response

Specify a valid value.

3054 (0BEE) (RC3054): MQRCCF_ESCAPE_TYPE_ERROR

Explanation

Escape type not valid.

The value specified for *EscapeType* is not valid.

Programmer response

Specify a valid value.

3062 (0BF6) (RC3062): MQRCCF_CHANNEL_TABLE_ERROR

Explanation

Channel table value not valid.

The *ChannelTable* specified was not valid, or was not appropriate for the channel type specified on an Inquire Channel or Inquire Channel Names command.

Programmer response

Specify a valid channel table value.

3063 (0BF7) (RC3063): MQRCCF_MCA_TYPE_ERROR

Explanation

Message channel agent type not valid.

The *MCAType* value specified was not valid.

Programmer response

Specify a valid value.

3064 (0BF8) (RC3064): MQRCCF_CHL_INST_TYPE_ERROR

Explanation

Channel instance type not valid.

The *ChannelInstanceType* specified was not valid.

Programmer response

Specify a valid channel instance type.

3065 (0BF9) (RC3065): MQRCCF_CHL_STATUS_NOT_FOUND

Explanation

Channel status not found.

For Inquire Channel Status, no channel status is available for the specified channel. This might indicate that the channel has not been used.

Programmer response

None, unless this is unexpected, in which case consult your systems administrator.

3066 (0BFA) (RC3066): MQRCCF_CFSL_DUPLICATE_PARM

Explanation

Duplicate parameter.

Two MQCFSL structures, or an MQCFST followed by an MQCFSL structure, with the same parameter identifier were present.

Programmer response

Check for and remove duplicate parameters.

3067 (0BFB) (RC3067): MQRCCF_CFSL_TOTAL_LENGTH_ERROR

Explanation

Total string length error.

The total length of the strings (not including trailing blanks) in a MQCFSL structure exceeds the maximum allowable for the parameter.

Programmer response

Check that the structure has been specified correctly, and if so reduce the number of strings.

3068 (0BFC) (RC3068): MQRCCF_CFSL_COUNT_ERROR

Explanation

Count of parameter values not valid.

The MQCFSL *Count* field value was not valid. The value was negative or greater than the maximum permitted for the parameter specified in the *Parameter* field.

Programmer response

Specify a valid count for the parameter.

3069 (0BFD) (RC3069): MQRCCF_CFSL_STRING_LENGTH_ERR

Explanation

String length not valid.

The MQCFSL *StringLength* field value was not valid. The value was negative or greater than the maximum permitted length of the parameter specified in the *Parameter* field.

Programmer response

Specify a valid string length for the parameter.

3070 (0BFE) (RC3070): MQRCCF_BROKER_DELETED

Explanation

Broker has been deleted.

When a broker is deleted using the `dltmqbrk` command, all broker queues created by the broker are deleted. Before this can be done the queues are emptied of all command messages; any that are found are placed on the dead-letter queue with this reason code.

Programmer response

Process the command messages that were placed on the dead-letter queue.

3071 (0BFF) (RC3071): MQRCCF_STREAM_ERROR

Explanation

Stream name is not valid.

The stream name parameter is not valid. Stream names must obey the same naming rules as for IBM MQ queues.

Programmer response

Retry the command with a valid stream name parameter.

3072 (0C00) (RC3072): MQRCCF_TOPIC_ERROR

Explanation

Topic name is invalid.

A command has been sent to the broker containing a topic name that is not valid. Note that wildcard topic names are not allowed for *Register Publisher* and *Publish* commands.

Programmer response

Retry the command with a valid topic name parameter. Up to 256 characters of the topic name in question are returned with the error response message. If the topic name contains a null character, this is assumed to terminate the string and is not considered to be part of it. A zero length topic name is not valid, as is one that contains an escape sequence that is not valid.

3073 (0C01) (RC3073): MQRCCF_NOT_REGISTERED

Explanation

Subscriber or publisher is not registered.

A *Deregister* command has been issued to remove registrations for a topic, or topics, for which the publisher or subscriber is not registered. If multiple topics were specified on the command, it fails with a completion code of MQCC_WARNING if the publisher or subscriber was registered for some, but not all, of the topics specified. This error code is also returned to a subscriber issuing a *Request Update* command for a topic for which he does not have a subscription.

Programmer response

Investigate why the publisher or subscriber is not registered. In the case of a subscriber, the subscriptions might have expired, or been removed automatically by the broker if the subscriber is no longer authorized.

3074 (0C02) (RC3074): MQRCCF_Q_MGR_NAME_ERROR

Explanation

An invalid or unknown queue manager name has been supplied.

A queue manager name has been supplied as part of a publisher or subscriber identity. This might have been supplied as an explicit parameter or in the *ReplyToQMgr* field in the message descriptor of the command. Either the queue manager name is not valid, or in the case of a subscriber identity, the subscriber's queue could not be resolved because the remote queue manager is not known to the broker queue manager.

Programmer response

Retry the command with a valid queue manager name. If appropriate, the broker includes a further error reason code within the error response message. If one is supplied, follow the guidance for that reason code in "Reason codes and exceptions" on page 1314 to resolve the problem.

3075 (0C03) (RC3075): MQRCCF_INCORRECT_STREAM

Explanation

Stream name does not match the stream queue it was sent to.

A command has been sent to a stream queue that specified a different stream name parameter.

Programmer response

Retry the command either by sending it to the correct stream queue or by modifying the command so that the stream name parameter matches.

3076 (0C04) (RC3076): MQRCCF_Q_NAME_ERROR

Explanation

An invalid or unknown queue name has been supplied.

A queue name has been supplied as part of a publisher or subscriber identity. This might have been supplied as an explicit parameter or in the *ReplyToQ* field in the message descriptor of the command. Either the queue name is not valid, or in the case of a subscriber identity, the broker has failed to open the queue.

Programmer response

Retry the command with a valid queue name. If appropriate, the broker includes a further error reason code within the error response message. If one is supplied, follow the guidance for that reason code in "Reason codes and exceptions" on page 1314 to resolve the problem.

3077 (0C05) (RC3077): MQRCCF_NO_RETAINED_MSG

Explanation

No retained message exists for the topic specified.

A *Request Update* command has been issued to request the retained message associated with the specified topic. No retained message exists for that topic.

Programmer response

If the topic or topics in question should have retained messages, the publishers of these topics might not be publishing with the correct publication options to cause their publications to be retained.

3078 (0C06) (RC3078): MQRCCF_DUPLICATE_IDENTITY

Explanation

Publisher or subscriber identity already assigned to another user ID.

Each publisher and subscriber has a unique identity consisting of a queue manager name, a queue name, and optionally a correlation identifier. Associated with each identity is the user ID under which that publisher or subscriber first registered. A specific identity can be assigned only to one user ID at a time. While the identity is registered with the broker all commands wanting to use it must specify the correct user ID. When a publisher or a subscriber no longer has any registrations with the broker the identity can be used by another user ID.

Programmer response

Either retry the command using a different identity or remove all registrations associated with the identity so that it can be used by a different user ID. The user ID to which the identity is currently assigned is returned within the error response message. A *Deregister* command could be issued to remove these registrations. If the user ID in question cannot be used to execute such a command, you need to have the necessary authority to open the SYSTEM.BROKER.CONTROL.QUEUE using the MQOO_ALTERNATE_USER_AUTHORITY option.

3079 (0C07) (RC3079): MQRCCF_INCORRECT_Q

Explanation

Command sent to wrong broker queue.

The command is a valid broker command but the queue it has been sent to is incorrect. `Publish` and `Delete Publication` commands need to be sent to the stream queue, all other commands need to be sent to the SYSTEM.BROKER.CONTROL.QUEUE.

Programmer response

Retry the command by sending it to the correct queue.

3080 (0C08) (RC3080): MQRCCF_CORREL_ID_ERROR

Explanation

Correlation identifier used as part of an identity is all binary zeros.

Each publisher and subscriber is identified by a queue manager name, a queue name, and optionally a correlation identifier. The correlation identifier is typically used to allow multiple subscribers to share the same subscriber queue. In this instance a publisher or subscriber has indicated within the Registration or Publication options supplied on the command that their identity does include a correlation identifier, but a valid identifier has not been supplied. The <RegOpt>CorrelAsId</RegOpt> has been specified, but the correlation identifier of the message is nulls.

Programmer response

Change the program to retry the command ensuring that the correlation identifier supplied in the message descriptor of the command message is not all binary zeros.

3081 (0C09) (RC3081): MQRCCF_NOT_AUTHORIZED

Explanation

Subscriber has insufficient authority.

To receive publications a subscriber application needs both browse authority for the stream queue that it is subscribing to, and put authority for the queue that publications are to be sent to. Subscriptions are rejected if the subscriber does not have both authorities. In addition to having browse authority for the stream queue, a subscriber would also require `altusr` authority for the stream queue to subscribe to certain topics that the broker itself publishes information on. These topics start with the MQ/SA/ prefix.

Programmer response

Ensure that the subscriber has the necessary authorities and reissue the request. The problem might occur because the subscriber's user ID is not known to the broker. This can be identified if a further error reason code of MQRC_UNKNOWN_ENTITY is returned within the error response message.

3082 (0C0A) (RC3082): MQRCCF_UNKNOWN_STREAM

Explanation

Stream is not known by the broker or could not be created.

A command message has been put to the SYSTEM.BROKER.CONTROL.QUEUE for an unknown stream. This error code is also returned if dynamic stream creation is enabled and the broker failed to create a stream queue for the new stream using the SYSTEM.BROKER.MODEL.STREAM queue.

Programmer response

Retry the command for a stream that the broker supports. If the broker should support the stream, either define the stream queue manually, or correct the problem that prevented the broker from creating the stream queue itself.

3083 (0C0B) (RC3083): MQRCCF_REG_OPTIONS_ERROR

Explanation

Invalid registration options have been supplied.

The registration options (between <RegOpt> and </RegOpt>) provided on a command are not valid.

Programmer response

Retry the command with a valid combination of options.

3084 (0C0C) (RC3084): MQRCCF_PUB_OPTIONS_ERROR

Explanation

Invalid publication options have been supplied.

The publication options provided on a Publish command are not valid.

Programmer response

Retry the command with a valid combination of options.

3085 (0C0D) (RC3085): MQRCCF_UNKNOWN_BROKER

Explanation

Command received from an unknown broker.

Within a multi-broker network, related brokers pass subscriptions and publications between each other as a series of command messages. One such command message has been received from a broker that is not, or is no longer, related to the detecting broker.

Programmer response

This situation can occur if the broker network is not quiesced while topology changes are made to the network.

If you are removing a broker from the topology when the queue manager is inactive, your changes are propagated at queue manager restart.

If you are removing a broker from the topology when the queue manager is active, make sure the channels are also active, so that your changes are immediately propagated.

3086 (0C0E) (RC3086): MQRCCF_Q_MGR_CCSID_ERROR

Explanation

Queue manager coded character set identifier error.

The coded character set value for the queue manager was not valid.

Programmer response

Specify a valid value.

3087 (0C0F) (RC3087): MQRCCF_DEL_OPTIONS_ERROR

Explanation

Invalid delete options have been supplied.

The options provided with a *Delete Publication* command are not valid.

Programmer response

Retry the command with a valid combination of options.

3088 (0C10) (RC3088): MQRCCF_CLUSTER_NAME_CONFLICT

Explanation

ClusterName and *ClusterNamelist* attributes conflict.

The command was rejected because it would have resulted in the *ClusterName* attribute and the *ClusterNamelist* attribute both being nonblank. At least one of these attributes must be blank.

Programmer response

If the command specified one of these attributes only, you must also specify the other one, but with a value of blanks. If the command specified both attributes, ensure that one of them has a value of blanks.

3089 (0C11) (RC3089): MQRCCF_REPOS_NAME_CONFLICT

Explanation

RepositoryName and *RepositoryNamelist* attributes conflict.

Either:
- The command was rejected because it would have resulted in the *RepositoryName* and *RepositoryNamelist* attributes both being nonblank. At least one of these attributes must be blank.
- For a Reset Queue Manager Cluster command, the queue manager does not provide a full repository management service for the specified cluster. That is, the *RepositoryName* attribute of the queue manager is not the specified cluster name, or the namelist specified by the *RepositoryNamelist* attribute does not contain the cluster name.

Programmer response

Reissue the command with the correct values or on the correct queue manager.

3090 (0C12) (RC3090): MQRCCF_CLUSTER_Q_USAGE_ERROR

Explanation

Queue cannot be a cluster queue.

The command was rejected because it would have resulted in a cluster queue also being a transmission queue, which is not permitted, or because the queue in question cannot be a cluster queue.

Programmer response

Ensure that the command specifies either:
- The *Usage* parameter with a value of MQUS_NORMAL, or
- The *ClusterName* and *ClusterNamelist* parameters with values of blanks.
- A *QName* parameter with a value that is not one of these reserved queues:
  - SYSTEM.CHANNEL.INITQ
  - SYSTEM.CHANNEL.SYNCQ
  - SYSTEM.CLUSTER.COMMAND.QUEUE
  - SYSTEM.CLUSTER.REPOSITORY.QUEUE
  - SYSTEM.COMMAND.INPUT
  - SYSTEM.QSG.CHANNEL.SYNCQ
  - SYSTEM.QSG.TRANSMIT.QUEUE

3091 (0C13) (RC3091): MQRCCF_ACTION_VALUE_ERROR

Explanation

Action value not valid.

The value specified for *Action* is not valid. There is only one valid value.

Programmer response

Specify MQACT_FORCE_REMOVE as the value of the *Action* parameter.

3092 (0C14) (RC3092): MQRCCF_COMMS_LIBRARY_ERROR

Explanation

Library for requested communications protocol could not be loaded.

The library needed for the requested communications protocol could not be loaded.

Programmer response

Install the library for the required communications protocol, or specify a communications protocol that has already been installed.

3093 (0C15) (RC3093): MQRCCF_NETBIOS_NAME_ERROR

Explanation

NetBIOS listener name not defined.

The NetBIOS listener name is not defined.

Programmer response

Add a local name to the configuration file and retry the operation.

3094 (0C16) (RC3094): MQRCCF_BROKER_COMMAND_FAILED

Explanation

The broker command failed to complete.

A broker command was issued but it failed to complete.

Programmer response

Diagnose the problem using the provided information and issue a corrected command.

For more information, look at the IBM MQ error logs.

3095 (0C17) (RC3095): MQRCCF_CFST_CONFLICTING_PARM

Explanation

Conflicting parameters.

The command was rejected because the parameter identified in the error response was in conflict with another parameter in the command.

Programmer response

Consult the description of the parameter identified to ascertain the nature of the conflict, and the correct command.

3096 (0C18) (RC3096): MQRCCF_PATH_NOT_VALID

Explanation

Path not valid.

The path specified was not valid.

Programmer response

Specify a valid path.

3097 (0C19) (RC3097): MQRCCF_PARM_SYNTAX_ERROR

Explanation

Syntax error found in parameter.

The parameter specified contained a syntax error.

Programmer response

Check the syntax for this parameter.

3098 (0C1A) (RC3098): MQRCCF_PWD_LENGTH_ERROR

Explanation

Password length error.

The password string length is rounded up by to the nearest eight bytes. This rounding causes the total length of the *SSLCryptoHardware* string to exceed its maximum.

Programmer response

Decrease the size of the password, or of earlier fields in the *SSLCryptoHardware* string.

3150 (0C4E) (RC3150): MQRCCF_FILTER_ERROR

Explanation

Filter not valid. This could be because either:

1. In an inquire command message, the specification of a filter is not valid.
2. In a publish/subscribe command message, the content-based filter expression supplied in the publish/subscribe command message contains invalid syntax, and cannot be used.

Programmer response

1. Correct the specification of the filter parameter structure in the inquire command message.
2. Correct the syntax of the filter expression in the publish/subscribe command message. The filter expression is the value of the `Filter` tag in the `psc` folder in the MQRFH2 structure. See the *Websphere MQ Integrator V2 Programming Guide* for details of valid syntax.

3151 (0C4F) (RC3151): MQRCCF_WRONG_USER

Explanation

Wrong user.

A publish/subscribe command message cannot be executed on behalf of the requesting user because the subscription that it would update is already owned by a different user. A subscription can be updated or deregistered only by the user that originally registered the subscription.

Programmer response

Ensure that applications that need to issue commands against existing subscriptions are running under the user identifier that originally registered the subscription. Alternatively, use different subscriptions for different users.

3152 (0C50) (RC3152): MQRCCF_DUPLICATE_SUBSCRIPTION

Explanation

The subscription already exists.

A matching subscription already exists.

Programmer response

Either modify the new subscription properties to distinguish it from the existing subscription or deregister the existing subscription. Then reissue the command.

3153 (0C51) (RC3153): MQRCCF_SUB_NAME_ERROR

Explanation

The subscription name parameter is in error.

Either the subscription name is of an invalid format or a matching subscription already exists with no subscription name.

Programmer response

Either correct the subscription name or remove it from the command and reissue the command.

3154 (0C52) (RC3154): MQRCCF_SUB_IDENTITY_ERROR

Explanation

The subscription identity parameter is in error.

Either the supplied value exceeds the maximum length allowed or the subscription identity is not currently a member of the subscription's identity set and a Join registration option was not specified.

Programmer response

Either correct the identity value or specify a Join registration option to add this identity to the identity set for this subscription.

3155 (0C53) (RC3155): MQRCCF_SUBSCRIPTION_IN_USE

Explanation

The subscription is in use.

An attempt to modify or deregister a subscription was attempted by a member of the identity set when they were not the only member of this set.

Programmer response

Reissue the command when you are the only member of the identity set. To avoid the identity set check and force the modification or deregistration remove the subscription identity from the command message and reissue the command.

3156 (0C54) (RC3156): MQRCCF_SUBSCRIPTION_LOCKED

Explanation

The subscription is locked.

The subscription is currently exclusively locked by another identity.

Programmer response

Wait for this identity to release the exclusive lock.

3157 (0C55) (RC3157): MQRCCF_ALREADY_JOINED

Explanation

The identity already has an entry for this subscription.

A Join registration option was specified but the subscriber identity was already a member of the subscription's identity set.

Programmer response

None. The command completed, this reason code is a warning.

3160 (0C58) (RC3160): MQRCCF_OBJECT_IN_USE

Explanation

Object in use by another command.

A modification of an object was attempted while the object was being modified by another command.

Programmer response

Retry the command.

3161 (0C59) (RC3161): MQRCCF_UNKNOWN_FILE_NAME

Explanation

File not defined to CICS.

A file name parameter identifies a file that is not defined to CICS.

Programmer response

Provide a valid file name or create a CSD definition for the required file.

3162 (0C5A) (RC3162): MQRCCF_FILE_NOT_AVAILABLE

Explanation

File not available to CICS.

A file name parameter identifies a file that is defined to CICS, but is not available.

Programmer response

Check that the CSD definition for the file is correct and enabled.

3163 (0C5B) (RC3163): MQRCCF_DISC_RETRY_ERROR

Explanation

Disconnection retry count not valid.

The *DiscRetryCount* value was not valid.

Programmer response

Specify a valid count.

3164 (0C5C) (RC3164): MQRCCF_ALLOC_RETRY_ERROR

Explanation

Allocation retry count not valid.

The *AllocRetryCount* value was not valid.

Programmer response

Specify a valid count.

3165 (0C5D) (RC3165): MQRCCF_ALLOC_SLOW_TIMER_ERROR

Explanation

Allocation slow retry timer value not valid.

The *AllocRetrySlowTimer* value was not valid.

Programmer response

Specify a valid timer value.

3166 (0C5E) (RC3166): MQRCCF_ALLOC_FAST_TIMER_ERROR

Explanation

Allocation fast retry timer value not valid.

The *AllocRetryFastTimer* value was not valid.

Programmer response

Specify a valid value.

3167 (0C5F) (RC3167): MQRCCF_PORT_NUMBER_ERROR

Explanation

Port number value not valid.

The *PortNumber* value was not valid.

Programmer response

Specify a valid port number value.

3168 (0C60) (RC3168): MQRCCF_CHL_SYSTEM_NOT_ACTIVE

Explanation

Channel system is not active.

An attempt was made to start a channel while the channel system was inactive.

Programmer response

Activate the channel system before starting a channel.

3169 (0C61) (RC3169): MQRCCF_ENTITY_NAME_MISSING

Explanation

Entity name required but missing.

A parameter specifying entity names must be supplied.

Programmer response

Specify the required parameter.

3170 (0C62) (RC3170): MQRCCF_PROFILE_NAME_ERROR

Explanation

Profile name not valid.

A profile name is not valid. Profile names might include wildcard characters or might be given explicitly. If you give an explicit profile name, then the object identified by the profile name must exist. This error might also occur if you specify more than one double asterisk in a profile name.

Programmer response

Specify a valid name.

3171 (0C63) (RC3171): MQRCCF_AUTH_VALUE_ERROR

Explanation

Authorization value not valid.

A value for the *AuthorizationList* or *AuthorityRemove* or *AuthorityAdd* parameter was not valid.

Programmer response

Specify a valid value.

3172 (0C64) (RC3172): MQRCCF_AUTH_VALUE_MISSING

Explanation

Authorization value required but missing.

A parameter specifying authorization values must be supplied.

Programmer response

Specify the required parameter.

3173 (0C65) (RC3173): MQRCCF_OBJECT_TYPE_MISSING

Explanation

Object type value required but missing.

A parameter specifying the object type must be supplied.

Programmer response

Specify the required parameter.

3174 (0C66) (RC3174): MQRCCF_CONNECTION_ID_ERROR

Explanation

Error in connection id parameter.

The `ConnectionId` specified was not valid.

Programmer response

Specify a valid connection id.

3175 (0C67) (RC3175): MQRCCF_LOG_TYPE_ERROR

Explanation

Log type not valid.

The log type value specified was not valid.

Programmer response

Specify a valid log type value.

3176 (0C68) (RC3176): MQRCCF_PROGRAM_NOT_AVAILABLE

Explanation

Program not available.

A request to start or stop a service failed because the request to start the program failed. This could be because the program could not be found at the specified location, or that insufficient system resources are available currently to start it.

Programmer response

Check that the correct name is specified in the definition of the service, and that the program is in the appropriate libraries, before retrying the request.

3177 (0C69) (RC3177): MQRCCF_PROGRAM_AUTH_FAILED

Explanation

Program not available.

A request to start or stop a service failed because the user does not have sufficient access authority to start the program at the specified location.

Programmer response

Correct the progam name and location, and the user's authority, before retrying the request.

3200 (0C80) (RC3200): MQRCCF_NONE_FOUND

Explanation

No items found matching request criteria.

An Inquire command found no items that matched the specified name and satisfied any other criteria requested.

3201 (0C81) (RC3201): MQRCCF_SECURITY_SWITCH_OFF

Explanation

Security refresh or reverification not processed, security switch set OFF.

Either
- a Reverify Security command was issued, but the subsystem security switch is off, so there are no internal control tables to flag for reverification; or
- a Refresh Security command was issued, but the security switch for the requested class or the subsystem security switch is off.

The switch in question might be returned in the message (with parameter identifier MQIACF_SECURITY_SWITCH).

3202 (0C82) (RC3202): MQRCCF_SECURITY_REFRESH_FAILED

Explanation

Security refresh did not take place.

A SAF RACROUTE REQUEST=STAT call to your external security manager (ESM) returned a non-zero return code. In consequence, the requested security refresh could not be done. The security item affected might be returned in the message (with parameter identifier MQIACF_SECURITY_ITEM).

Possible causes of this problem are:
- The class is not installed
- The class is not active
- The external security manager (ESM) is not active
- The RACF z/OS router table is incorrect

Programmer response

For information about resolving the problem, see the explanations of messages CSQH003I and CSQH004I.

3203 (0C83) (RC3203): MQRCCF_PARM_CONFLICT

Explanation

Incompatible parameters or parameter values.

The parameters or parameter values for a command are incompatible. One of the following occurred:
- A parameter was not specified that is required by another parameter or parameter value.
- A parameter or parameter value was specified that is not allowed with some other parameter or parameter value.
- The values for two specified parameters were not both blank or non-blank.
- The values for two specified parameters were incompatible.

The parameters in question might be returned in the message (with parameter identifiers MQIACF_PARAMETER_ID).

Programmer response

Reissue the command with correct parameters and values.

3204 (0C84) (RC3204): MQRCCF_COMMAND_INHIBITED

Explanation

Commands not allowed at present time.

The queue manager cannot accept commands at the present time, because it is restarting or terminating, or because the command server is not running.

3205 (0C85) (RC3205): MQRCCF_OBJECT_BEING_DELETED

Explanation

Object is being deleted.

The object specified on a command is in the process of being deleted, so the command is ignored.

3207 (0C87) (RC3207): MQRCCF_STORAGE_CLASS_IN_USE

Explanation

Storage class is active or queue is in use.

The command for a local queue involved a change to the *StorageClass* value, but there are messages on the queue, or other threads have the queue open.

Programmer response

Remove the messages from the queue, or wait until any other threads have closed the queue.

3208 (0C88) (RC3208): MQRCCF_OBJECT_NAME_RESTRICTED

Explanation

Incompatible object name and type.

The command used a reserved object name with an incorrect object type or subtype. The object is only allowed to be of a predetermined type, as listed in the explanation of message CSQM108I.

3209 (0C89) (RC3209): MQRCCF_OBJECT_LIMIT_EXCEEDED

Explanation

Local queue limit exceeded.

The command failed because no more local queues could be defined. There is an implementation limit of 524 287 for the total number of local queues that can exist. For shared queues, there is a limit of 512 queues in a single coupling facility structure.

Programmer response

Delete any existing queues that are no longer required.

3210 (0C8A) (RC3210): MQRCCF_OBJECT_OPEN_FORCE

Explanation

Object is in use, but could be changed specifying *Force* as MQFC_YES.

The object specified is in use. This could be because it is open through the API, or for certain parameter changes, because there are messages currently on the queue. The requested changes can be made by specifying *Force* as MQFC_YES on a Change command.

Programmer response

Wait until the object is not in use. Alternatively specify *Force* as MQFC_YES for a change command.

3211 (0C8B) (RC3211): MQRCCF_DISPOSITION_CONFLICT

Explanation

Parameters are incompatible with disposition.

The parameters or parameter values for a command are incompatible with the disposition of an object. One of the following occurred:
- A value specified for the object name or other parameter is not allowed for a local queue with a disposition that is shared or a model queue used to create a dynamic queue that is shared.
- A value specified for a parameter is not allowed for an object with such disposition.
- A value specified for a parameter must be non-blank for an object with such disposition.
- The *CommandScope* and *QSGDisposition* or *ChannelDisposition* parameter values are incompatible.
- The action requested for a channel cannot be performed because it has the wrong disposition.

The parameter and disposition in question may be returned in the message (with parameter identifiers MQIACF_PARAMETER_ID and MQIA_QSG_DISP).

Programmer response

Reissue the command with correct parameters and values.

3212 (0C8C) (RC3212): MQRCCF_Q_MGR_NOT_IN_QSG

Explanation

Queue manager is not in a queue-sharing group.

The command or its parameters are not allowed when the queue manager is not in a queue-sharing group. The parameter in question might be returned in the message (with parameter identifier MQIACF_PARAMETER_ID).

Programmer response

Reissue the command correctly.

3213 (0C8D) (RC3213): MQRCCF_ATTR_VALUE_FIXED

Explanation

Parameter value cannot be changed.

The value for a parameter cannot be changed. The parameter in question might be returned in the message (with parameter identifier MQIACF_PARAMETER_ID).

Programmer response

To change the parameter, the object must be deleted and then created again with the new value.

3215 (0C8F) (RC3215): MQRCCF_NAMELIST_ERROR

Explanation

Namelist is empty or wrong type.

A namelist used to specify a list of clusters has no names in it or does not have type MQNT_CLUSTER or MQNT_NONE.

Programmer response

Reissue the command specifying a namelist that is not empty and has a suitable type.

3217 (0C91) (RC3217): MQRCCF_NO_CHANNEL_INITIATOR

Explanation

Channel initiator not active.

The command requires the channel initiator to be started.

3218 (0C93) (RC3218): MQRCCF_CHANNEL_INITIATOR_ERROR

Explanation

Channel initiator cannot be started, or no suitable channel initiator is available.

This might occur because of the following reasons:
- The channel initiator cannot be started because:
  - It is already active.
  - There are insufficient system resources.
  - The queue manager was shutting down.
- The shared channel cannot be started because there was no suitable channel initiator available for any active queue manager in the queue-sharing group. This could be because:
  - No channel initiators are running.
  - The channel initiators that are running are too busy to allow any channel, or a channel of the particular type, to be started.

3222 (0C96) (RC3222): MQRCCF_COMMAND_LEVEL_CONFLICT

Explanation

Incompatible queue manager command levels.

Changing the *CFLevel* parameter of a CF structure, or deleting a CF structure, requires that all queue managers in the queue-sharing group have a command level of at least 530. Some of the queue managers have a level less than 530.

3223 (0C97) (RC3223): MQRCCF_Q_ATTR_CONFLICT

Explanation

Queue attributes are incompatible.

The queues involved in a Move Queue command have different values for one or more of these attributes: *DefinitionType*, *HardenGetBackout*, *Usage*. Messages cannot be moved safely if these attributes differ.

3224 (0C98) (RC3224): MQRCCF_EVENTS_DISABLED

Explanation

Events not enabled.

The command required performance or configuration events to be enabled.

Programmer response

Use the Change Queue manager command to enable the events if required.

3225 (0C99) (RC3225): MQRCCF_COMMAND_SCOPE_ERROR

Explanation

Queue-sharing group error.

While processing a command that used the *CommandScope* parameter, an error occurred while trying to send data to the coupling facility.

Programmer response

Notify your system programmer.

3226 (0C9A) (RC3226): MQRCCF_COMMAND_REPLY_ERROR

Explanation

Error saving command reply information.

While processing a command that used the *CommandScope* parameter, or a command for the channel initiator, an error occurred while trying to save information about the command.

Programmer response

The most likely cause is insufficient storage. If the problem persists, you may need to restart the queue manager after making more storage available.

3227 (0C9B) (RC3227): MQRCCF_FUNCTION_RESTRICTED

Explanation

Restricted command or parameter value used.

The command, or the value specified for one of its parameters, is not allowed because the installation and customization options chosen do not allow all functions to be used. The parameter in question might be returned in the message (with parameter identifier MQIACF_PARAMETER_ID).

3228 (0C9C) (RC3228): MQRCCF_PARM_MISSING

Explanation

Required parameter not specified.

The command did not specify a parameter or parameter value that was required. It might be for one of the following reasons:
- A parameter that is always required.
- A parameter that is one of a set of two or more alternative required parameters.
- A parameter that is required because some other parameter was specified.
- A parameter that is a list of values which has too few values.

The parameter in question might be returned in the message (with parameter identifier MQIACF_PARAMETER_ID).

Programmer response

Reissue the command with correct parameters and values.

3229 (0C9D) (RC3229): MQRCCF_PARM_VALUE_ERROR

Explanation

Parameter value invalid.

The value specified for a parameter was not acceptable. It might be for one of the following reasons:
- Outside the acceptable numeric range for the parameter.
- Not one of a list of acceptable values for the parameter.
- Using characters that are invalid for the parameter.
- Completely blank, when such is not allowed for the parameter.
- A filter value that is invalid for the parameter being filtered.

The parameter in question might be returned in the message (with parameter identifier MQIACF_PARAMETER_ID).

Programmer response

Reissue the command with correct parameters and values.

3230 (0C9E) (RC3230): MQRCCF_COMMAND_LENGTH_ERROR

Explanation

Command exceeds allowable length.

The command is so large that its internal form has exceeded the maximum length allowed. The size of the internal form of the command is affected by both the length, and the complexity of the command.

3231 (0C9F) (RC3231): MQRCCF_COMMAND_ORIGIN_ERROR

Explanation

Command issued incorrectly.

The command cannot be issued using command server. This is an internal error.

Programmer response

Notify your system programmer.

3232 (0CA0) (RC3232): MQRCCF_LISTENER_CONFLICT

Explanation

Address conflict for listener.

A listener was already active for a port and IP address combination that conflicted with the *Port* and *IPAddress* values specified by a Start Channel Listener or Stop Channel Listener command. The *Port* and *IPAddress* value combination specified must match a combination for which the listener is active. It cannot be a superset or a subset of that combination.

Programmer response

Reissue the command with correct values, if required.

3233 (0CA1) (RC3233): MQRCCF_LISTENER_STARTED

Explanation

Listener is started.

An attempt was made to start a listener, but it is already active for the requested *TransportType*, *InboundDisposition*, *Port*, and *IPAddress* values. The requested parameter values might be returned in the message, if applicable (with parameter identifiers MQIACH_XMIT_PROTOCOL_TYPE, MQIACH_INBOUND_DISP, MQIACH_PORT_NUMBER, MQCACH_IP_ADDRESS).

3234 (0CA2) (RC3234): MQRCCF_LISTENER_STOPPED

Explanation

Listener is stopped.

An attempt was made to stop a listener, but it is not active or already stopping for the requested `TransportType`, `InboundDisposition`, `Port`, and `IPAddress` values. The requested parameter values might be returned in the message, if applicable (with parameter identifiers MQIACH_XMIT_PROTOCOL_TYPE, MQIACH_INBOUND_DISP, MQIACH_PORT_NUMBER, MQCACH_IP_ADDRESS).

3235 (0CA3) (RC3235): MQRCCF_CHANNEL_ERROR

Explanation

Channel command failed.

A channel command failed because of an error in the channel definition, or at the remote end of the channel, or in the communications system. An error identifier value *nnn* may be returned in the message (with parameter identifier MQIACF_ERROR_ID).

Programmer response

For information about the error, see the explanation of the corresponding error message. Error *nnn* generally corresponds to message CSQX *nnn*, although there are some exceptions. z/OS For more information, see Distributed queuing message codes.

3236 (0CA4) (RC3236): MQRCCF_CF_STRUC_ERROR

Explanation

CF structure error.

A command could not be processed because of a coupling facility or CF structure error. It might be:
- A Backup CF Structure or Recover CF Structure command when the status of the CF structure is unsuitable. In this case, the CF structure status might be returned in the message together with the CF structure name (with parameter identifiers MQIACF_CF_STRUC_STATUS and MQCA_CF_STRUC_NAME).
- A command could not access an object because of an error in the coupling facility information, or because a CF structure has failed. In this case, the name of the object involved might be returned in the message (with parameter identifier MQCA_Q_NAME, for example).
- A command involving a shared channel could not access the channel status or synchronization key information.

Programmer response

In the case of a Backup CF Structure or Recover CF Structure command, take action appropriate to the CF structure status reported.

In other cases, check for error messages on the console log that might relate to the problem. Check whether the coupling facility structure has failed and check that Db2 is available.

3237 (0CA5) (RC3237): MQRCCF_UNKNOWN_USER_ID

Explanation

User identifier not found.

A user identifier specified in a Reverify Security command was not valid because there was no entry found for it in the internal control table. This could be because the identifier was entered incorrectly in the command, or because it was not in the table (for example, because it had timed-out). The user identifier in question might be returned in the message (with parameter identifier MQCACF_USER_IDENTIFIER).

3238 (0CA6) (RC3238): MQRCCF_UNEXPECTED_ERROR

Explanation

Unexpected or severe error.

An unexpected or severe error or other failure occurred. A code associated with the error might be returned in the message (with parameter identifier MQIACF_ERROR_ID).

Programmer response

Notify your system programmer.

3239 (0CA7) (RC3239): MQRCCF_NO_XCF_PARTNER

Explanation

MQ is not connected to the XCF partner.

The command involving the IMS bridge cannot be processed because MQ is not connected to the XCF partner. The group and member names of the XCF partner in question might be returned in the message (with parameter identifiers MQCA_XCF_GROUP_NAME and MQCA_XCF_MEMBER_NAME).

3240 (0CA8) (RC3240): MQRCCF_CFGR_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFGR `Parameter` field value was not valid.

Programmer response

Specify a valid parameter identifier.

3241 (0CA9) (RC3241): MQRCCF_CFIF_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFIF *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3242 (0CAA) (RC3242): MQRCCF_CFIF_OPERATOR_ERROR

Explanation

Parameter count not valid.

The MQCFIF *Operator* field value was not valid.

Programmer response

Specify a valid operator value.

3243 (0CAB) (RC3243): MQRCCF_CFIF_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFIF *Parameter* field value was not valid, or specifies a parameter that cannot be filtered, or that is also specified as a parameter to select a subset of objects.

Programmer response

Specify a valid parameter identifier.

3244 (0CAC) (RC3244): MQRCCF_CFSF_FILTER_VAL_LEN_ERR

Explanation

Filter value length not valid.

The MQCFSF *FilterValueLength* field value was not valid.

Programmer response

Specify a valid length.

3245 (0CAD) (RC3245): MQRCCF_CFSF_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFSF *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3246 (0CAE) (RC3246): MQRCCF_CFSF_OPERATOR_ERROR

Explanation

Parameter count not valid.

The MQCFSF *Operator* field value was not valid.

Programmer response

Specify a valid operator value.

3247 (0CAF) (RC3247): MQRCCF_CFSF_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFSF *Parameter* field value was not valid.

Programmer response

Specify a valid parameter identifier.

3248 (0CB0) (RC3248): MQRCCF_TOO_MANY_FILTERS

Explanation

Too many filters.

The command contained more than the maximum permitted number of filter structures.

Programmer response

Specify the command correctly.

3249 (0CB1) (RC3249): MQRCCF_LISTENER_RUNNING

Explanation

Listener is running.

An attempt was made to perform an operation on a listener, but it is currently active.

Programmer response

Stop the listener if required.

3250 (0CB2) (RC3250): MQRCCF_LSTR_STATUS_NOT_FOUND

Explanation

Listener status not found.

For Inquire Listener Status, no listener status is available for the specified listener. This might indicate that the listener has not been used.

Programmer response

None, unless this is unexpected, in which case consult your systems administrator.

3251 (0CB3) (RC3251): MQRCCF_SERVICE_RUNNING

Explanation

Service is running.

An attempt was made to perform an operation on a service, but it is currently active.

Programmer response

Stop the service if required.

3252 (0CB4) (RC3252): MQRCCF_SERV_STATUS_NOT_FOUND

Explanation

Service status not found.

For Inquire Service Status, no service status is available for the specified service. This might indicate that the service has not been used.

Programmer response

None, unless this is unexpected, in which case consult your systems administrator.

3253 (0CB5) (RC3253): MQRCCF_SERVICE_STOPPED

Explanation

Service is stopped.

An attempt was made to stop a service, but it is not active or already stopping.

3254 (0CB6) (RC3254): MQRCCF_CFBS_DUPLICATE_PARM

Explanation

Duplicate parameter.

Two MQCFBS structures with the same parameter identifier were present.

Programmer response

Check for and remove duplicate parameters.

3255 (0CB7) (RC3255): MQRCCF_CFBS_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFBS *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3256 (0CB8) (RC3256): MQRCCF_CFBS_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFBS *Parameter* field value was not valid.

Programmer response

Specify a valid parameter identifier.

3257 (0CB9) (RC3257): MQRCCF_CFBS_STRING_LENGTH_ERR

Explanation

String length not valid.

The MQCFBS *StringLength* field value was not valid. The value was negative or greater than the maximum permitted length of the parameter specified in the *Parameter* field.

Programmer response

Specify a valid string length for the parameter.

3258 (0CBA) (RC3258): MQRCCF_CFGR_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFGR *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3259 (0CBB) (RC3259): MQRCCF_CFGR_PARM_COUNT_ERROR

Explanation

Parameter count not valid.

The MQCFGR *ParameterCount* field value was not valid. The value was negative or greater than the maximum permitted for the parameter identifier specified in the *Parameter* field.

Programmer response

Specify a valid count for the parameter.

3260 (0CBC) (RC3260): MQRCCF_CONN_NOT_STOPPED

Explanation

Connection not stopped.

The Stop Connection command could not be executed, so the connection was not stopped.

3261 (0CBD) (RC3261): MQRCCF_SERVICE_REQUEST_PENDING

Explanation

A Suspend or Resume Queue Manager command was issued, or a Refresh Security command, but such a command is currently in progress.

Programmer response

Wait until the current request completes, then reissue the command if necessary.

3262 (0CBE) (RC3262): MQRCCF_NO_START_CMD

Explanation

No start command.

The service cannot be started because no start command is specified in the service definition.

Programmer response

Correct the definition of the service.

3263 (0CBF) (RC3263): MQRCCF_NO_STOP_CMD

Explanation

No stop command.

The service cannot be stopped because no stop command is specified in the service definition.

Programmer response

Correct the definition of the service.

3264 (0CC0) (RC3264): MQRCCF_CFBF_LENGTH_ERROR

Explanation

Structure length not valid.

The MQCFBF *StrucLength* field value was not valid.

Programmer response

Specify a valid structure length.

3265 (0CC1) (RC3265): MQRCCF_CFBF_PARM_ID_ERROR

Explanation

Parameter identifier is not valid.

The MQCFBF *Parameter* field value was not valid.

Programmer response

Specify a valid parameter identifier.

3266 (0CC2) (RC3266): MQRCCF_CFBF_FILTER_VAL_LEN_ERR

Explanation

Filter value length not valid.

The MQCFBF *FilterValueLength* field value was not valid.

Programmer response

Specify a valid length.

3267 (0CC3) (RC3267): MQRCCF_CFBF_OPERATOR_ERROR

Explanation

Parameter count not valid.

The MQCFBF *Operator* field value was not valid.

Programmer response

Specify a valid operator value.

3268 (0CC4) (RC3268): MQRCCF_LISTENER_STILL_ACTIVE

Explanation

Listener still active.

An attempt was made to stop a listener, but it failed and the listener is still active. For example, the listener might still have active channels.

Programmer response

Wait for the active connections to the listener to complete before trying the request again.

3269 (0CC5) (RC3269): MQRCCF_DEF_XMIT_Q_CLUS_ERROR

Explanation

The specified queue is not allowed to be used as the default transmission queue because it is reserved for use exclusively by clustering.

Programmer response

Change the value of the Default Transmission Queue, and try the command again.

3300 (0CE4) (RC3300): MQRCCF_TOPICSTR_ALREADY_EXISTS

Explanation

The topic string specified already exists in another topic object.

Programmer response

Verify that the topic string used is correct.

3301 (0CE5) (RC3301): MQRCCF_SHARING_CONVS_ERROR

Explanation

An invalid value has been given for SharingConversations parameter in the Channel definition

Programmer response

Correct the value used in the PCF SharingConversations (MQCFIN) parameter; see Change, Copy, and Create Channel for more information.

3302 (0CE6) (RC3302): MQRCCF_SHARING_CONVS_TYPE

Explanation

SharingConversations parameter is not allowed for this channel type.

Programmer response

See Change, Copy, and Create Channel to ensure that the channel type is compatible with the SharingConversations parameter.

3303 (0CE7) (RC3303): MQRCCF_SECURITY_CASE_CONFLICT

Explanation

A Refresh Security PCF command was issued, but the case currently in use differs from the system setting and if refreshed would result in the set of classes using different case settings.

Programmer response

Check that the class used is set up correctly and that the system setting is correct. If a change in case setting is required, issue the REFRESH SECURITY(*) command to change all classes.

3305 (0CE9) (RC3305): MQRCCF_TOPIC_TYPE_ERROR

Explanation

An Inquire or Delete Topic PCF command was issued with an invalid TopicType parameter.

Programmer response

Correct the TopicType parameter and reissue the command. For more details on the TopicType, see Change, Copy, and Create Topic.

3306 (0CEA) (RC3306): MQRCCF_MAX_INSTANCES_ERROR

Explanation

An invalid value was given for the maximum number of simultaneous instances of a server-connection channel (MaxInstances) for the channel definition.

Programmer response

See Change, Copy, and Create Channel for more information and correct the PCF application.

3307 (0CEB) (RC3307): MQRCCF_MAX_INSTS_PER_CLNT_ERR

Explanation

An invalid value was given for the MaxInstancesPerClient property.

Programmer response

See Change, Copy, and Create Channel for the range of values and correct the application.

3308 (0CEC) (RC3308): MQRCCF_TOPIC_STRING_NOT_FOUND

Explanation

When processing an Inquire Topic Status command, the topic string specified did not match any topic nodes in the topic tree.

Programmer response

Verify the topic string is correct.

3309 (0CED) (RC3309): MQRCCF_SUBSCRIPTION_POINT_ERR

Explanation

The Subscription point was not valid. Valid subscription points are the topic strings of the topic objects listed in the SYSTEM.QPUBSUB.SUBPOINT.NAMELIST.

Programmer response

Use a subscription point that matches the topic string of a topic object listed in the SYSTEM.QPUBSUB.SUBPOINT.NAMELIST (or remove the subscription point parameter and this uses the default subscription point)

3311 (0CEF) (RC2432): MQRCCF_SUB_ALREADY_EXISTS

Explanation

When processing a Copy or Create Subscription command, the target *Subscription* identifier exists.

Programmer response

If you are trying to copy an existing subscription, ensure that the *ToSubscriptionName* parameter contains a unique value. If you are trying to create a Subscription ensure that the combination of the *SubName* parameter, and *TopicObject* parameter or *TopicString* parameter are unique.

3314 (0CF2) (RC3314): MQRCCF_DURABILITY_NOT_ALLOWED

Explanation

An MQSUB call using the MQSO_DURABLE option failed. This can be for one of the following reasons:
- The topic subscribed to is defined as DURSUB(NO).
- The queue named SYSTEM.DURABLE.SUBSCRIBER.QUEUE is not available.
- The topic subscribed to is defined as both MCAST(ONLY) and DURSUB(YES) (or DURSUB(ASPARENT) and the parent is DURSUB(YES)).

Completion Code

MQCC_FAILED

Programmer Response

Durable subscriptions are stored on the SYSTEM.DURABLE.SUBSCRIBER.QUEUE. Ensure that this queue is available for use. Possible reasons for failure include the queue being full, the queue being put inhibited, the queue not existing, or (on z/OS) the pageset the queue is defined to use doesn't exist.

If the topic subscribed to is defined as DURSUB(NO) either alter the administrative topic node to use DURSUB(YES) or use the MQSO_NON_DURABLE option instead.

If the topic subscribed to is defined as MCAST(ONLY) when using IBM MQ Multicast messaging, alter the topic to use DURSUB(NO).

3317 (0CF5) (RC3317): MQRCCF_INVALID_DESTINATION

Explanation

The Subscription or Topic object used in a Change, Copy, Create or Delete PCF command is invalid.

Programmer response

Investigate and correct the required parameters for the specific command you are using. For more details, see Change, Copy, and Create Subscription.

3318 (0CF6) (RC3318): MQRCCF_PUBSUB_INHIBITED

Explanation

MQSUB, MQOPEN, MQPUT and MQPUT1 calls are currently inhibited for all publish/subscribe topics, either by means of the queue manager attribute PSMODE or because processing of publish/subscribe state at queue manager start-up has failed, or has not yet completed.

Completion Code

MQCC_FAILED

Programmer response

If this queue manager does not intentionally inhibit publish/subscribe, investigate any error messages that describe the failure at queue manager start-up, or wait until start-up processing completes. You can use the DISPLAY PUBSUB command to check the status of the publish/subscribe engine to ensure it is ready for use, and additionally on z/OS you will receive an information message CSQM076I.

3326 (0CFE) (RC3326): MQRCCF_CHLAUTH_TYPE_ERROR

Explanation

Channel authentication record type not valid.

The **`type`** parameter specified on the **`set`** command was not valid.

Programmer response

Specify a valid type.

3327 (0CFF) (RC3327): MQRCCF_CHLAUTH_ACTION_ERROR

Explanation

Channel authentication record action not valid.

The **`action`** parameter specified on the **`set`** command was not valid.

Programmer response

Specify a valid action.

3335 (0D07) (RC3335): MQRCCF_CHLAUTH_USRSRC_ERROR

Explanation

Channel authentication record user source not valid.

The **`user source`** parameter specified on the **`set`** command was not valid.

Programmer response

Specify a valid user source.

3336 (0D08) (RC3336): MQRCCF_WRONG_CHLAUTH_TYPE
Explanation

Parameter not allowed for this channel authentication record type.

The parameter is not allowed for the type of channel authentication record being set. Refer to the description of the parameter in error to determine the types of record for which this parameter is valid.

Programmer response

Remove the parameter.

3337 (0D09) (RC3337): MQRCCF_CHLAUTH_ALREADY_EXISTS
Explanation

Channel authentication record already exists

An attempt was made to add a channel authentication record, but it already exists.

Programmer response

Specify action as MQACT_REPLACE.

3338 (0D0A) (RC3338): MQRCCF_CHLAUTH_NOT_FOUND
Explanation

Channel authentication record not found.

The specified channel authentication record does not exist.

Programmer response

Specify a channel authentication record that exists.

3339 (0D0B) (RC3339): MQRCCF_WRONG_CHLAUTH_ACTION
Explanation

Parameter not allowed for this action on a channel authentication record.

The parameter is not allowed for the action being applied to a channel authentication record. Refer to the description of the parameter in error to determine the actions for which this parameter is valid.

Programmer response

Remove the parameter.

3340 (0D0C) (RC3340): MQRCCF_WRONG_CHLAUTH_USERSRC

Explanation

Parameter not allowed for this channel authentication record user source value.

The parameter is not allowed for a channel authentication record with the value that the **`user source`** field contains. Refer to the description of the parameter in error to determine the values of user source for which this parameter is valid.

Programmer response

Remove the parameter.

3341 (0D0D) (RC3341): MQRCCF_CHLAUTH_WARN_ERROR

Explanation

Channel authentication record **`warn`** value not valid.

The **`warn`** parameter specified on the **`set`** command was not valid.

Programmer response

Specify a valid value for **`warn`**.

3342 (0D0E) (RC3342): MQRCCF_WRONG_CHLAUTH_MATCH

Explanation

Parameter not allowed for this channel authentication record **`match`** value.

The parameter is not allowed for an **`inquire channel authentication record`** command with the value that the **`match`** field contains. Refer to the description of the parameter in error to find the values of **`match`** for which this parameter is valid.

Programmer response

Remove the parameter.

3343 (0D0F) (RC3343): MQRCCF_IPADDR_RANGE_CONFLICT

Explanation

A channel authentication record contained an IP address with a range that overlapped an existing range. A range must be a superset or subset of any existing ranges for the same channel profile name, or completely separate.

Programmer response

Specify a range that is a superset or subset of an existing range, or is completely separate to all existing ranges.

3344 (0D10) (RC3344): MQRCCF_CHLAUTH_MAX_EXCEEDED
Explanation

A channel authentication record was set taking the total number of entries for that type on a single channel profile over the maximum number allowed.

Programmer response

Remove some channel authentication records to make room.

3345 (0D11) (RC3345): MQRCCF_IPADDR_ERROR
Explanation

A channel authentication record contained an invalid IP address, or invalid wildcard pattern to match against IP addresses.

Programmer response

Specify a valid IP address or pattern.

Related information:

Generic IP addresses

3346 (0D12) (RC3346): MQRCCF_IPADDR_RANGE_ERROR
Explanation

A channel authentication record contained an IP address with a range that was invalid, for example, the lower number is higher than or equal to the upper number of the range.

Programmer response

Specify a valid range in the IP address.

3347 (0D13) (RC3347): MQRCCF_PROFILE_NAME_MISSING

Explanation

Profile name missing.

A profile name was required for the command but none was specified.

Programmer response

Specify a valid profile name.

3348 (0D14) (RC3348): MQRCCF_CHLAUTH_CLNTUSER_ERROR
Explanation

Channel authentication record **`client user`** value not valid.

The **`client user`** value contains a wildcard character, which is not allowed.

Programmer response

Specify a valid value for the client user field.

3349 (0D15) (RC3349): MQRCCF_CHLAUTH_NAME_ERROR
Explanation

Channel authentication record channel name not valid.

When a channel authentication record specifies an IP address to block, the **`channel name`** value must be a single asterisk (*).

Programmer response

Enter a single asterisk in the channel name.

3350 (0D16) (RC3350): MQRCCF_CHLAUTH_RUNCHECK_ERROR
Runcheck command is using generic values.

Explanation

An Inquire Channel Authentication Record command using MQMATCH_RUNCHECK was issued, but one or more of the input fields on the command were provided with generic values, which is not allowed.

Programmer response

Enter non-generic values for channel name, address, one of the client user ID or remote queue manager and SSL Peer Name if used.

3353 (0D19) (RC3353): MQRCCF_SUITE_B_ERROR
Invalid values have been specified.

Explanation

An invalid combination of values has been specified for the **`MQIA_SUITE_B_STRENGTH`** parameter.

Programmer response

Review the combination entered and retry with appropriate values.

3363 (0D23) (RC3363): MQRCCF_CLUS_XMIT_Q_USAGE_ERROR

Explanation

If the local queue attribute **CLCHNAME** is set, the attribute **USAGE** must be set to XMITQ.

On z/OS, if the local queue attribute **CLCHNAME** is set, the attribute **INDXTYPE** must be set to **CORRELID**, and the transmission queue must not be a shared queue.

The **CLCHNAME** attribute is a generic cluster-sender channel name. It identifies the cluster-sender channel that transfers messages in a transmission queue to another queue manager.

Programmer response

Modify the application to set the **CLCHNAME** to blanks, or not set the **CLCHNAME** attribute at all, on queues other than transmission queues.

On z/OS, ensure that the transmission queue is indexed by correlation ID and the queue is not a shared queue.

3364 (0D24) (RC3364): MQRCCF_CERT_VAL_POLICY_ERROR

Explanation

An invalid certificate validation policy value was specified for the **MQIA_CERT_VAL_POLICY** attribute. The specified value is unknown or is not supported on the current platform.

Programmer response

Review the value specified and try again with an appropriate certificate validation policy.

3366 (0D26) (RC3366): MQRCCF_REVDNS_DISABLED

Explanation

A runcheck command completed successfully returning the records to be used. However, some Channel Authentication Records exist containing hostnames, and hostname reverse lookup is currently disabled, so these records will not have been matched against. This reason code is returned as an MQCC_WARNING.

Programmer response

If reverse lookup is correctly disabled, even though some Channel Authentication Records exist containing hostnames, this warning can be ignored.

If channel authentication records containing hostnames should be being matched against, and therefore reverse lookup of hostname should not currently be disabled, issue a Change Queue Manager command to re-enable it.

If reverse lookup for hostnames is correctly disabled and there should not be any Channel Authentication Records containing hostnames, issue a Set Channel Authentication Record to remove them.

3370 (0D2A) (RC3370): MQRCCF_CHLAUTH_CHKCLI_ERROR

Explanation

Channel authentication record check client not valid.

The check client parameter specified on the set command was not valid.

Programmer response

Specify a valid user source.

3377 (0D31) (RC3377): MQRCCF_TOPIC_RESTRICTED

V 8.0.0.2

Explanation

This error can occur when creating or modifying a topic object. One or more attributes of the topic object are not supported on an IBM MQ administrative topic.

Programmer response

Modify the configuration to adhere to the documented restrictions.

4001 (0FA1) (RC4001): MQRCCF_OBJECT_ALREADY_EXISTS

Explanation

Object already exists.

An attempt was made to create an object, but the object already existed and the *Replace* parameter was not specified as MQRP_YES.

Programmer response

Specify *Replace* as MQRP_YES, or use a different name for the object to be created.

4002 (0FA2) (RC4002): MQRCCF_OBJECT_WRONG_TYPE

Explanation

Object has wrong type or disposition.

An object already exists with the same name but a different subtype or disposition from that specified by the command.

Programmer response

Ensure that the specified object is the same subtype and disposition.

4003 (0FA3) (RC4003): MQRCCF_LIKE_OBJECT_WRONG_TYPE

Explanation

New and existing objects have different subtype.

An attempt was made to create an object based on the definition of an existing object, but the new and existing objects had different subtypes.

Programmer response

Ensure that the new object has the same subtype as the one on which it is based.

4004 (0FA4) (RC4004): MQRCCF_OBJECT_OPEN

Explanation

Object is open.

An attempt was made to operate on an object that was in use.

Programmer response

Wait until the object is not in use, and then retry the operation. Alternatively specify *Force* as MQFC_YES for a change command.

4005 (0FA5) (RC4005): MQRCCF_ATTR_VALUE_ERROR

Explanation

Attribute value not valid or repeated.

One or more of the attribute values specified are not valid or are repeated. The error response message contains the failing attribute selectors (with parameter identifier MQIACF_PARAMETER_ID).

Programmer response

Specify the attribute values correctly.

4006 (0FA6) (RC4006): MQRCCF_UNKNOWN_Q_MGR

Explanation

Queue manager not known.

The queue manager specified was not known.

Programmer response

Specify the name of the queue manager to which the command is sent, or blank.

4007 (0FA7) (RC4007): MQRCCF_Q_WRONG_TYPE

Explanation

Action not valid for the queue of specified type.

An attempt was made to perform an action on a queue of the wrong type.

Programmer response

Specify a queue of the correct type.

4008 (0FA8) (RC4008): MQRCCF_OBJECT_NAME_ERROR

Explanation

Name not valid.

An object or other name name was specified using characters that were not valid.

Programmer response

Specify only valid characters for the name.

4009 (0FA9) (RC4009): MQRCCF_ALLOCATE_FAILED

Explanation

Allocation failed.

An attempt to allocate a conversation to a remote system failed. The error might be due to an entry in the channel definition that is not valid, or it might be that the listening program at the remote system is not running.

Programmer response

Ensure that the channel definition is correct, and start the listening program if necessary. If the error persists, consult your systems administrator.

4010 (0FAA) (RC4010): MQRCCF_HOST_NOT_AVAILABLE

Explanation

Remote system not available.

An attempt to allocate a conversation to a remote system was unsuccessful. The error might be transitory, and the allocate might succeed later. This reason can occur if the listening program at the remote system is not running.

Programmer response

Ensure that the listening program is running, and retry the operation.

4011 (0FAB) (RC4011): MQRCCF_CONFIGURATION_ERROR

Explanation

Configuration error.

There was a configuration error in the channel definition or communication subsystem, and allocation of a conversation was not possible. This might be caused by one of the following:

- For LU 6.2, either the *ModeName* or the *TpName* is incorrect. The *ModeName* must match that on the remote system, and the *TpName* must be specified. (On IBM i, these are held in the communications Side Object.)
- For LU 6.2, the session might not be established.
- For TCP, the *ConnectionName* in the channel definition cannot be resolved to a network address. This might be because the name has not been correctly specified, or because the name server is not available.
- The requested communications protocol might not be supported on the platform.

Programmer response

Identify the error and take appropriate action.

4012 (0FAC) (RC4012): MQRCCF_CONNECTION_REFUSED

Explanation

Connection refused.

The attempt to establish a connection to a remote system was rejected. The remote system might not be configured to allow a connection from this system.

- For LU 6.2 either the user ID or the password supplied to the remote system is incorrect.
- For TCP the remote system might not recognize the local system as valid, or the TCP listener program might not be started.

Programmer response

Correct the error or restart the listener program.

4013 (0FAD) (RC4013): MQRCCF_ENTRY_ERROR

Explanation

Connection name not valid.

The connection name in the channel definition could not be resolved into a network address. Either the name server does not contain the entry, or the name server was not available.

Programmer response

Ensure that the connection name is correctly specified and that the name server is available.

4014 (0FAE) (RC4014): MQRCCF_SEND_FAILED

Explanation

Send failed.

An error occurred while sending data to a remote system. This might be caused by a communications failure.

Programmer response

Consult your systems administrator.

4015 (0FAF) (RC4015): MQRCCF_RECEIVED_DATA_ERROR

Explanation

Received data error.

An error occurred while receiving data from a remote system. This might be caused by a communications failure.

Programmer response

Consult your systems administrator.

4016 (0FB0) (RC4016): MQRCCF_RECEIVE_FAILED

Explanation

Receive failed.

The receive operation failed.

Programmer response

Correct the error and retry the operation.

4017 (0FB1) (RC4017): MQRCCF_CONNECTION_CLOSED

Explanation

Connection closed.

An error occurred while receiving data from a remote system. The connection to the remote system has unexpectedly terminated.

Programmer response

Contact your systems administrator.

4018 (0FB2) (RC4018): MQRCCF_NO_STORAGE

Explanation

Not enough storage available.

Insufficient storage is available.

Programmer response

Consult your systems administrator.

4019 (0FB3) (RC4019): MQRCCF_NO_COMMS_MANAGER

Explanation

Communications manager not available.

The communications subsystem is not available.

Programmer response

Ensure that the communications subsystem has been started.

4020 (0FB4) (RC4020): MQRCCF_LISTENER_NOT_STARTED

Explanation

Listener not started.

The listener program could not be started. Either the communications subsystem has not been started, or the number of current channels using the communications subsystem is the maximum allowed, or there are too many jobs waiting in the queue.

Programmer response

Ensure the communications subsystem is started or retry the operation later. Increase the number of current channels allowed, if appropriate.

4024 (0FB8) (RC4024): MQRCCF_BIND_FAILED

Explanation

Bind failed.

The bind to a remote system during session negotiation has failed.

Programmer response

Consult your systems administrator.

4025 (0FB9) (RC4025): MQRCCF_CHANNEL_INDOUBT

Explanation

Channel in-doubt.

The requested operation cannot complete because the channel is in doubt.

Programmer response

Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or resolve the channel.

4026 (0FBA) (RC4026): MQRCCF_MQCONN_FAILED

Explanation

MQCONN call failed.

Programmer response

Check whether the queue manager is active.

4027 (0FBB) (RC4027): MQRCCF_MQOPEN_FAILED

Explanation

MQOPEN call failed.

Programmer response

Check whether the queue manager is active, and the queues involved are correctly set up.

4028 (0FBC) (RC4028): MQRCCF_MQGET_FAILED

Explanation

MQGET call failed.

Programmer response

Check whether the queue manager is active, and the queues involved are correctly set up, and enabled for MQGET.

4029 (0FBD) (RC4029): MQRCCF_MQPUT_FAILED

Explanation

MQPUT call failed.

Programmer response

Check whether the queue manager is active, and the queues involved are correctly set up, and not inhibited for puts.

4030 (0FBE) (RC4030): MQRCCF_PING_ERROR

Explanation

Ping error.

A ping operation can only be issued for a sender or server channel. If the local channel is a receiver channel, you must issue the ping from a remote queue manager.

Programmer response

Reissue the ping request for a different channel of the correct type, or for a receiver channel from a different queue manager.

4031 (0FBF) (RC4031): MQRCCF_CHANNEL_IN_USE

Explanation

Channel in use.

An attempt was made to perform an operation on a channel, but the channel is currently active.

Programmer response

Stop the channel or wait for it to terminate.

4032 (0FC0) (RC4032): MQRCCF_CHANNEL_NOT_FOUND

Explanation

Channel not found.

The channel specified does not exist.

Programmer response

Specify the name of a channel which exists.

4033 (0FC1) (RC4033): MQRCCF_UNKNOWN_REMOTE_CHANNEL

Explanation

Remote channel not known.

There is no definition of the referenced channel at the remote system.

Programmer response

Ensure that the local channel is correctly defined. If it is, add an appropriate channel definition at the remote system.

4034 (0FC2) (RC4034): MQRCCF_REMOTE_QM_UNAVAILABLE

Explanation

Remote queue manager not available.

The channel cannot be started because the remote queue manager is not available.

Programmer response

Start the remote queue manager.

4035 (0FC3) (RC4035): MQRCCF_REMOTE_QM_TERMINATING

Explanation

Remote queue manager terminating.

The channel is ending because the remote queue manager is terminating.

Programmer response

Restart the remote queue manager.

4036 (0FC4) (RC4036): MQRCCF_MQINQ_FAILED

Explanation

MQINQ call failed.

Programmer response

Check whether the queue manager is active.

4037 (0FC5) (RC4037): MQRCCF_NOT_XMIT_Q

Explanation

Queue is not a transmission queue.

The queue specified in the channel definition is not a transmission queue, or is in use.

Programmer response

Ensure that the queue is specified correctly in the channel definition, and that it is correctly defined to the queue manager.

4038 (0FC6) (RC4038): MQRCCF_CHANNEL_DISABLED

Explanation

Channel disabled.

An attempt was made to use a channel, but the channel was disabled (that is, stopped).

Programmer response

Start the channel.

4039 (0FC7) (RC4039): MQRCCF_USER_EXIT_NOT_AVAILABLE

Explanation

User exit not available.

The channel was terminated because the user exit specified does not exist.

Programmer response

Ensure that the user exit is correctly specified and the program is available.

4040 (0FC8) (RC4040): MQRCCF_COMMIT_FAILED

Explanation

Commit failed.

An error was received when an attempt was made to commit a unit of work.

Programmer response

Consult your systems administrator.

4041 (0FC9) (RC4041): MQRCCF_WRONG_CHANNEL_TYPE

Explanation

Parameter not allowed for this channel type.

The parameter is not allowed for the type of channel being created, copied, or changed. Refer to the description of the parameter in error to determine the types of channel for which the parameter is valid

Programmer response

Remove the parameter.

4042 (0FCA) (RC4042): MQRCCF_CHANNEL_ALREADY_EXISTS

Explanation

Channel already exists.

An attempt was made to create a channel but the channel already existed and *Replace* was not specified as MQRP_YES.

Programmer response

Specify *Replace* as MQRP_YES or use a different name for the channel to be created.

4043 (0FCB) (RC4043): MQRCCF_DATA_TOO_LARGE

Explanation

Data too large.

The data to be sent exceeds the maximum that can be supported for the command.

Programmer response

Reduce the size of the data.

4044 (0FCC) (RC4044): MQRCCF_CHANNEL_NAME_ERROR

Explanation

Channel name error.

The *ChannelName* parameter contained characters that are not allowed for channel names.

Programmer response

Specify a valid name.

4045 (0FCD) (RC4045): MQRCCF_XMIT_Q_NAME_ERROR

Explanation

Transmission queue name error.

The *XmitQName* parameter contains characters that are not allowed for queue names. This reason code also occurs if the parameter is not present when a sender or server channel is being created, and no default value is available.

Programmer response

Specify a valid name, or add the parameter.

4047 (0FCF) (RC4047): MQRCCF_MCA_NAME_ERROR

Explanation

Message channel agent name error.

The *MCAName* value contained characters that are not allowed for program names on the platform in question.

Programmer response

Specify a valid name.

4048 (0FD0) (RC4048): MQRCCF_SEND_EXIT_NAME_ERROR

Explanation

Channel send exit name error.

The *SendExit* value contained characters that are not allowed for program names on the platform in question.

Programmer response

Specify a valid name.

4049 (0FD1) (RC4049): MQRCCF_SEC_EXIT_NAME_ERROR

Explanation

Channel security exit name error.

The *SecurityExit* value contained characters that are not allowed for program names on the platform in question.

Programmer response

Specify a valid name.

4050 (0FD2) (RC4050): MQRCCF_MSG_EXIT_NAME_ERROR

Explanation

Channel message exit name error.

The *MsgExit* value contained characters that are not allowed for program names on the platform in question.

Programmer response

Specify a valid name.

4051 (0FD3) (RC4051): MQRCCF_RCV_EXIT_NAME_ERROR

Explanation

Channel receive exit name error.

The *ReceiveExit* value contained characters that are not allowed for program names on the platform in question.

Programmer response

Specify a valid name.

4052 (0FD4) (RC4052): MQRCCF_XMIT_Q_NAME_WRONG_TYPE

Explanation

Transmission queue name not allowed for this channel type.

The *XmitQName* parameter is only allowed for sender or server channel types.

Programmer response

Remove the parameter.

4053 (0FD5) (RC4053): MQRCCF_MCA_NAME_WRONG_TYPE

Explanation

Message channel agent name not allowed for this channel type.

The *MCAName* parameter is only allowed for sender, server or requester channel types.

Programmer response

Remove the parameter.

4054 (0FD6) (RC4054): MQRCCF_DISC_INT_WRONG_TYPE

Explanation

Disconnection interval not allowed for this channel type.

The *DiscInterval* parameter is only allowed for sender or server channel types.

Programmer response

Remove the parameter.

4055 (0FD7) (RC4055): MQRCCF_SHORT_RETRY_WRONG_TYPE

Explanation

Short retry parameter not allowed for this channel type.

The *ShortRetryCount* parameter is only allowed for sender or server channel types.

Programmer response

Remove the parameter.

4056 (0FD8) (RC4056): MQRCCF_SHORT_TIMER_WRONG_TYPE

Explanation

Short timer parameter not allowed for this channel type.

The *ShortRetryInterval* parameter is only allowed for sender or server channel types.

Programmer response

Remove the parameter.

4057 (0FD9) (RC4057): MQRCCF_LONG_RETRY_WRONG_TYPE

Explanation

Long retry parameter not allowed for this channel type.

The *LongRetryCount* parameter is only allowed for sender or server channel types.

Programmer response

Remove the parameter.

4058 (0FDA) (RC4058): MQRCCF_LONG_TIMER_WRONG_TYPE

Explanation

Long timer parameter not allowed for this channel type.

The *LongRetryInterval* parameter is only allowed for sender or server channel types.

Programmer response

Remove the parameter.

4059 (0FDB) (RC4059): MQRCCF_PUT_AUTH_WRONG_TYPE

Explanation

Put authority parameter not allowed for this channel type.

The *PutAuthority* parameter is only allowed for receiver or requester channel types.

Programmer response

Remove the parameter.

4061 (0FDD) (RC4061): MQRCCF_MISSING_CONN_NAME

Explanation

Connection name parameter required but missing.

The *ConnectionName* parameter is required for sender or requester channel types, but is not present.

Programmer response

Add the parameter.

4062 (0FDE) (RC4062): MQRCCF_CONN_NAME_ERROR

Explanation

Error in connection name parameter.

The *ConnectionName* parameter contains one or more blanks at the start of the name.

Programmer response

Specify a valid connection name.

4063 (0FDF) (RC4063): MQRCCF_MQSET_FAILED

Explanation

MQSET call failed.

Programmer response

Check whether the queue manager is active.

4064 (0FE0) (RC4064): MQRCCF_CHANNEL_NOT_ACTIVE

Explanation

Channel not active.

An attempt was made to stop a channel, but the channel was already stopped.

Programmer response

No action is required.

4065 (0FE1) (RC4065): MQRCCF_TERMINATED_BY_SEC_EXIT

Explanation

Channel terminated by security exit.

A channel security exit terminated the channel.

Programmer response

Check that the channel is attempting to connect to the correct queue manager, and if so that the security exit is specified correctly, and is working correctly, at both ends.

4067 (0FE3) (RC4067): MQRCCF_DYNAMIC_Q_SCOPE_ERROR

Explanation

Dynamic queue scope error.

The *Scope* attribute of the queue is to be MQSCO_CELL, but this is not allowed for a dynamic queue.

Programmer response

Predefine the queue if it is to have cell scope.

4068 (0FE4) (RC4068): MQRCCF_CELL_DIR_NOT_AVAILABLE

Explanation

Cell directory is not available.

The *Scope* attribute of the queue is to be MQSCO_CELL, but no name service supporting a cell directory has been configured.

Programmer response

Configure the queue manager with a suitable name service.

4069 (0FE5) (RC4069): MQRCCF_MR_COUNT_ERROR

Explanation

Message retry count not valid.

The *MsgRetryCount* value was not valid.

Programmer response

Specify a value in the range 0-999 999 999.

4070 (0FE6) (RC4070): MQRCCF_MR_COUNT_WRONG_TYPE

Explanation

Message-retry count parameter not allowed for this channel type.

The *MsgRetryCount* parameter is allowed only for receiver and requester channels.

Programmer response

Remove the parameter.

4071 (0FE7) (RC4071): MQRCCF_MR_EXIT_NAME_ERROR

Explanation

Channel message-retry exit name error.

The *MsgRetryExit* value contained characters that are not allowed for program names on the platform in question.

Programmer response

Specify a valid name.

4072 (0FE8) (RC4072): MQRCCF_MR_EXIT_NAME_WRONG_TYPE

Explanation

Message-retry exit parameter not allowed for this channel type.

The *MsgRetryExit* parameter is allowed only for receiver and requester channels.

Programmer response

Remove the parameter.

4073 (0FE9) (RC4073): MQRCCF_MR_INTERVAL_ERROR

Explanation

Message retry interval not valid.

The *MsgRetryInterval* value was not valid.

Programmer response

Specify a value in the range 0-999 999 999.

4074 (0FEA) (RC4074): MQRCCF_MR_INTERVAL_WRONG_TYPE

Explanation

Message-retry interval parameter not allowed for this channel type.

The *MsgRetryInterval* parameter is allowed only for receiver and requester channels.

Programmer response

Remove the parameter.

4075 (0FEB) (RC4075): MQRCCF_NPM_SPEED_ERROR

Explanation

Nonpersistent message speed not valid.

The *NonPersistentMsgSpeed* value was not valid.

Programmer response

Specify MQNPMS_NORMAL or MQNPMS_FAST.

4076 (0FEC) (RC4076): MQRCCF_NPM_SPEED_WRONG_TYPE

Explanation

Nonpersistent message speed parameter not allowed for this channel type.

The *NonPersistentMsgSpeed* parameter is allowed only for sender, receiver, server, requester, cluster sender, and cluster receiver channels.

Programmer response

Remove the parameter.

4077 (0FED) (RC4077): MQRCCF_HB_INTERVAL_ERROR

Explanation

Heartbeat interval not valid.

The *HeartbeatInterval* value was not valid.

Programmer response

Specify a value in the range 0-999 999.

4078 (0FEE) (RC4078): MQRCCF_HB_INTERVAL_WRONG_TYPE

Explanation

Heartbeat interval parameter not allowed for this channel type.

The *HeartbeatInterval* parameter is allowed only for receiver and requester channels.

Programmer response

Remove the parameter.

4079 (0FEF) (RC4079): MQRCCF_CHAD_ERROR

Explanation

Channel automatic definition error.

The *ChannelAutoDef* value was not valid.

Programmer response

Specify MQCHAD_ENABLED or MQCHAD_DISABLED.

4080 (0FF0) (RC4080): MQRCCF_CHAD_WRONG_TYPE

Explanation

Channel automatic definition parameter not allowed for this channel type.

The *ChannelAutoDef* parameter is allowed only for receiver and server-connection channels.

Programmer response

Remove the parameter.

4081 (0FF1) (RC4081): MQRCCF_CHAD_EVENT_ERROR

Explanation

Channel automatic definition event error.

The *ChannelAutoDefEvent* value was not valid.

Programmer response

Specify MQEVR_ENABLED or MQEVR_DISABLED.

4082 (0FF2) (RC4082): MQRCCF_CHAD_EVENT_WRONG_TYPE

Explanation

Channel automatic definition event parameter not allowed for this channel type.

The *ChannelAutoDefEvent* parameter is allowed only for receiver and server-connection channels.

Programmer response

Remove the parameter.

4083 (0FF3) (RC4083): MQRCCF_CHAD_EXIT_ERROR

Explanation

Channel automatic definition exit name error.

The *ChannelAutoDefExit* value contained characters that are not allowed for program names on the platform in question.

Programmer response

Specify a valid name.

4084 (0FF4) (RC4084): MQRCCF_CHAD_EXIT_WRONG_TYPE

Explanation

Channel automatic definition exit parameter not allowed for this channel type.

The *ChannelAutoDefExit* parameter is allowed only for receiver and server-connection channels.

Programmer response

Remove the parameter.

4085 (0FF5) (RC4085): MQRCCF_SUPPRESSED_BY_EXIT

Explanation

Action suppressed by exit program.

An attempt was made to define a channel automatically, but this was inhibited by the channel automatic definition exit. The *AuxErrorDataInt1* parameter contains the feedback code from the exit indicating why it inhibited the channel definition.

Programmer response

Examine the value of the *AuxErrorDataInt1* parameter, and take any action that is appropriate.

4086 (0FF6) (RC4086): MQRCCF_BATCH_INT_ERROR

Explanation

Batch interval not valid.

The batch interval specified was not valid.

Programmer response

Specify a valid batch interval value.

4087 (0FF7) (RC4087): MQRCCF_BATCH_INT_WRONG_TYPE

Explanation

Batch interval parameter not allowed for this channel type.

The *BatchInterval* parameter is allowed only for sender and server channels.

Programmer response

Remove the parameter.

4088 (0FF8) (RC4088): MQRCCF_NET_PRIORITY_ERROR

Explanation

Network priority value is not valid.

Programmer response

Specify a valid value.

4089 (0FF9) (RC4089): MQRCCF_NET_PRIORITY_WRONG_TYPE

Explanation

Network priority parameter not allowed for this channel type.

The *NetworkPriority* parameter is allowed for sender and server channels only.

Programmer response

Remove the parameter.

4090 (0FFA) (RC4090): MQRCCF_CHANNEL_CLOSED

Explanation

Channel closed.

The channel was closed prematurely. This can occur because a user stopped the channel while it was running, or a channel exit decided to close the channel.

Programmer response

Determine the reason that the channel was closed prematurely. Restart the channel if required.

4092 (0FFC) (RC4092): MQRCCF_SSL_CIPHER_SPEC_ERROR

Explanation

SSL cipher specification not valid.

The *SSLCipherSpec* specified is not valid.

Programmer response

Specify a valid cipher specification.

4093 (0FFD) (RC4093): MQRCCF_SSL_PEER_NAME_ERROR

Explanation

SSL peer name not valid.

The *SSLPeerName* specified is not valid.

Programmer response

Specify a valid peer name.

4094 (0FFE) (RC4094): MQRCCF_SSL_CLIENT_AUTH_ERROR

Explanation

SSL client authentication not valid.

The *SSLClientAuth* specified is not valid.

Programmer response

Specify a valid client authentication.

4095 (0FFF) (RC4095): MQRCCF_RETAINED_NOT_SUPPORTED

Explanation

Retained messages used on restricted stream.

An attempt has been made to use retained messages on a publish/subscribe stream defined to be restricted to JMS usage. JMS does not support the concept of retained messages and the request is rejected.

Programmer response

Either modify the application not to use retained messages, or modify the broker *JmsStreamPrefix* configuration parameter so that this stream is not treated as a JMS stream.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes

IBM MQ can use Secure Sockets Layer (SSL) with the various communication protocols. Use this topic to identify the error codes that can be returned by SSL.

The table in this appendix documents the return codes, in decimal form, from the Secure Sockets Layer (SSL) that can be returned in messages from the distributed queuing component.

If the return code is not listed, or if you want more information, see the *IBM Global Security Kit return codes* here: ../../SSPREK_6.1.0/com.ibm.itame.doc_6.1/am61_messages25.htm.

Table 146. SSL return codes

Return code (decimal)	Explanation
1	Handle is not valid.
3	An internal error has occurred.
4	Insufficient storage is available
5	Handle is in the incorrect state.
6	Key label is not found.
7	No certificates available.
8	Certificate validation error.
9	Cryptographic processing error.
10	ASN processing error.
11	LDAP processing error.
12	An unexpected error has occurred.
102	Error detected while reading key database or SAF key ring.
103	Incorrect key database record format.
106	Incorrect key database password.
109	No certificate authority certificates.
201	No key database password supplied.
202	Error detected while opening the key database.
203	Unable to generate temporary key pair
204	Key database password is expired.
302	Connection is active.
401	Certificate is expired or is not valid yet.
402	No SSL cipher specifications.
403	No certificate received from partner.
405	Certificate format is not supported.
406	Error while reading or writing data.
407	Key label does not exist.
408	Key database password is not correct.
410	SSL message format is incorrect.
411	Message authentication code is incorrect.
412	SSL protocol or certificate type is not supported.
413	Certificate signature is incorrect.
414	Certificate is not valid.
415	SSL protocol violation.
416	Permission denied.
417	Self-signed certificate cannot be validated.
420	Socket closed by remote partner.
421	SSL V2 cipher is not valid.
422	SSL V3 cipher is not valid.
427	LDAP is not available.

Table 146. SSL return codes (continued)

Return code (decimal)	Explanation
428	Key entry does not contain a private key.
429	SSL V2 header is not valid.
431	Certificate is revoked.
432	Session renegotiation is not allowed.
433	Key exceeds allowable export size.
434	Certificate key is not compatible with cipher suite.
435	Certificate authority is unknown.
436	Certificate revocation list cannot be processed.
437	Connection closed.
438	Internal error reported by remote partner.
439	Unknown alert received from remote partner.
501	Buffer size is not valid.
502	Socket request would block.
503	Socket read request would block.
504	Socket write request would block.
505	Record overflow.
601	Protocol is not SSL V3 or TLS V1.
602	Function identifier is not valid.
701	Attribute identifier is not valid.
702	The attribute has a negative length, which is invalid.
703	The enumeration value is invalid for the specified enumeration type.
704	Invalid parameter list for replacing the SID cache routines.
705	The value is not a valid number.
706	Conflicting parameters were set for additional certificate validation
707	The AES cryptographic algorithm is not supported.
708	The PEERID does not have the correct length.
1501	GSK_SC_OK
1502	GSK_SC_CANCEL
1601	The trace started successfully.
1602	The trace stopped successfully.
1603	No trace file was previously started so it cannot be stopped.
1604	Trace file already started so it cannot be started again.
1605	Trace file cannot be opened. The first parameter of gsk_start_trace() must be a valid full path filename.

In some cases, the secure sockets library reports a certificate validation error in an AMQ9633 error message. Table 2 lists the certificate validation errors that can be returned in messages from the distributed queuing component.

Table 147. Certificate validation errors.

A table listing return codes and explanations for certificate validation errors that can be returned in messages from the distributed queuing component.

Return code (decimal)	Explanation
575001	Internal error
575002	ASN error due to a malformed certificate
575003	Cryptographic error
575004	Key database error
575005	Directory error
575006	Invalid implementation library
575008	No appropriate validator
575009	The root CA is not trusted
575010	No certificate chain was built
575011	Digital signature algorithm mismatch
575012	Digital signature mismatch
575013	X.509 version does not allow Key IDs
575014	X.509 version does not allow extensions
575015	Unknown X.509 certificate version
575016	The certificate validity range is invalid
575017	The certificate is not yet valid
575018	The certificate has expired
575019	The certificate contains unknown critical extensions
575020	The certificate contains duplicate extensions
575021	The issuers directory name does not match the issuer's issuer
575022	The Authority Key ID serial number value does not match the serial number of the issuer
575023	The Authority Key ID and Subject Key ID do not match
575024	Unrecognized issuer alternative name
575025	The certificate Basic Constraints forbid use as a CA
575026	The certificate has a non-zero Basic Constraints path length but is not a CA
575027	The certificate Basic Constraints maximum path length was exceeded
575028	The certificate is not permitted to sign other certificates
575029	The certificate is not signed by a CA
575030	Unrecognized Subject Alternative Name
575031	The certificate chain is invalid
575032	The certificate is revoked
575033	Unrecognized CRL distribution point
575034	Name chaining failed
575035	Certificate is not in a chain
575036	The CRL is not yet valid
575037	The CRL has expired
575038	The certificate version does not allow critical extensions

Table 147. Certificate validation errors (continued).

A table listing return codes and explanations for certificate validation errors that can be returned in messages from the distributed queuing component.

Return code (decimal)	Explanation
575039	Unknown CRL distribution points
575040	No CRLs for CRL distribution points
575041	Indirect CRLs are not supported
575042	Missing issuing CRL distribution point name
575043	Distribution points do not match
575044	No available CRL data source
575045	CA Subject name is null
575046	Distinguished names do not chain
575047	Missing Subject Alternative Name
575048	Unique ID mismatch
575049	Name not permitted
575050	Name excluded
575051	CA certificate is missing Critical Basic Constraints
575052	Name constraints are not critical
575053	Name constraints minimum subtree value if set is not zero
575054	Name constraints maximum subtree value if set is not allowed
575055	Unsupported name constraint
575056	Empty policy constraints
575057	Bad certificate policies
575058	Certificate policies not acceptable
575059	Bad acceptable certificate policies
575060	Certificate policy mappings are critical
575061	Revocation status could not be determined
575062	Extended key usage error
575063	Unknown OCSP version
575064	Unknown OCSP response
575065	Bad OCSP key usage extension
575066	Bad OCSP nonce
575067	Missing OCSP nonce
575068	No OCSP client available

Related reference:

"API completion and reason codes" on page 1314
For each call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

"PCF reason codes" on page 1541
Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

"WCF custom channel exceptions"
Diagnostic messages are listed in this topic in numeric order, grouped according to the part of the WCF custom channel from which they originate.

Related information:

Diagnostic messages: AMQ4000-9999

z/OS IBM MQ for z/OS messages, completion, and reason codes

WCF custom channel exceptions

Diagnostic messages are listed in this topic in numeric order, grouped according to the part of the WCF custom channel from which they originate.

Reading a message

For each message, this information is provided:

- The message identifier, in two parts:
  1. The characters "WCFCH" which identify the message as being from the WCF custom channel for IBM MQ
  2. A four-digit decimal code followed by the character 'E'
- The text of the message.
- An explanation of the message giving further information.
- The response required from the user. In some cases, particularly for information messages, the response required might be "none".

Message variables

Some messages display text or numbers that vary according to the circumstances causing the message to occur; these circumstances are known as *message variables*. The message variables are indicated as {0}, {1}, and so on.

In some cases a message might have variables in the Explanation or Response. Find the values of the message variables by looking in the error log. The complete message, including the Explanation and the Response, is recorded there.

The following message types are described:

Related reference:

"API completion and reason codes" on page 1314
For each call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

"PCF reason codes" on page 1541
Reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message.

"Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes" on page 1622
IBM MQ can use Secure Sockets Layer (SSL) with the various communication protocols. Use this topic to identify the error codes that can be returned by SSL.

"WCF custom channel exceptions" on page 1627
Diagnostic messages are listed in this topic in numeric order, grouped according to the part of the WCF custom channel from which they originate.

Related information:

IBM MQ AMQ messages

z/OS IBM MQ for z/OS messages, completion, and reason codes

WCFCH0001E-0100E: General/State messages

Use the following information to understand WCFCH0001E-0100E general/state messages.

WCFCH0001E
An object cannot be opened because its state is '{0}'.

Explanation
An internal error has occurred.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM MQ support web page, or the IBM SupportAssistant web page, to see if a solution is already available. If you cannot find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

WCFCH0002E
An object cannot be closed because its state is '{0}'.

Explanation
An internal error has occurred.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM MQ support web page, or the IBM SupportAssistant web page, to see if a solution is already available. If you cannot find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

WCFCH0003E
An object cannot be used because its state is '{0}'.

Explanation
An internal error has occurred.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM MQ support web page, or the IBM SupportAssistant web page, to see if a solution is already available. If you cannot find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

WCFCH0004E
The specified 'Timeout' value '{0}' is out of range.

Explanation
The value is out of range, it must be greater than or equal to 'TimeSpan.Zero'.

Response
Specify a value which is in range or, to disable Timeout, specify a 'TimeSpan.MaxValue' value.

WCFCH0005E
The operation did not complete within the specified time of '{0}' for endpoint address '{1}'.

Explanation
A timeout occurred.

Response
Investigate the cause for the timeout.

WCFCH0006E
The parameter '{0}' is not of the expected type '{1}'

Explanation
A parameter with an unexpected type has been passed to a method call.

Response
Review the exception stack trace for further information.

WCFCH0007E
The parameter '{0}' must not be null.

Explanation
A method has been called with a required parameter set to a null value.

Response
Modify the application to provide a value for this parameter.

WCFCH0008E
An error occurred while processing an operation for endpoint address '{0}'.

Explanation
The operation failed to complete.

Response
Review the linked exceptions and stack trace for further information.

WCFCH0101E-0200E: URI Properties messages

Use the following information to understand WCFCH0101E-0200E URI properties messages.

WCFCH0101E
The endpoint URI must start with the valid character string '{0}'.

Explanation
The endpoint URI is incorrect, it must start with a valid character string.

Response
Specify an endpoint URI which starts with a valid character string.

WCFCH0102E
The endpoint URI must contain a '{0}' parameter with a value.

Explanation
The endpoint URI is incorrect, a parameter and its value are missing.

Response
Specify an endpoint URI with a value for this parameter.

WCFCH0103E
The endpoint URI must contain a '{0}' parameter with a value of '{1}'.

Explanation
The endpoint URI is incorrect, the parameter must contain the correct value.

Response
Specify an endpoint URI with a correct parameter and value.

WCFCH0104E
The endpoint URI contains a '{0}' parameter with an invalid value of '{1}'.

Explanation
The endpoint URI is incorrect, a valid parameter value must be specified.

Response
Specify an endpoint URI with a correct value for this parameter.

WCFCH0105E
The endpoint URI contains a '{0}' parameter with an invalid queue or queue manager name.

Explanation
The endpoint URI is incorrect, a valid queue and queue manager name must be specified.

Response
Specify an endpoint URI with valid values for the queue and the queue manager.

WCFCH0106E
The '{0}' property is a required property and must appear as the first property in the endpoint URI.

Explanation
The endpoint URI is incorrect, a parameter is either missing or in the wrong position.

Response
Specify an endpoint URI which contains this property as the first parameter.

WCFCH0107E
The property '{1}' cannot be used when the binding property is set to '{0}'.

Explanation
The endpoint URI connectionFactory parameter is incorrect, an invalid combination of properties has been used.

Response
Specify an endpoint URI connectionFactory which contains a valid combination of properties or binding.

WCFCH0109E
Property '{1}' must also be specified when property '{0}' is specified.

Explanation
The endpoint URI connectionFactory parameter is incorrect, it contains an invalid combination of properties.

Response
Specify an endpoint URI connectionFactory which contains a valid combination of properties.

WCFCH0110E
Property '{0}' has an invalid value '{1}'.

Explanation
The endpoint URI connectionFactory parameter is incorrect, the property does not contain a valid value.

Response
Specify an endpoint URI connectionFactory which contains a valid value for the property.

WCFCH0111E
: The value '{0}' is not supported for the binding mode property. XA operations are not supported.

Explanation
: The endpoint URI connectionFactory parameter is incorrect, the binding mode is not supported.

Response
: Specify an endpoint URI connectionFactory which contains a valid value for the binding mode.

WCFCH0112E
: The endpoint URI '{0}' is badly formatted.

Explanation
: The endpoint URI must follow the format described in the documentation.

Response
: Review the endpoint URI to ensure that it contains a valid value.

WCFCH0201E-0300E: Factory/Listener messages

Use the following information to understand WCFCH0201E-0300E factory/listener messages.

WCFCH0201E
: Channel shape '{0}' is not supported.

Explanation
: The users application or the WCF service contract has requested a channel shape which is not supported.

Response
: Identify and use a channel shape which is supported by the channel.

WCFCH0202E
: '{0}' MessageEncodingBindingElements have been specified.

Explanation
: The WCF binding configuration used by an application contains more than one message encoder.

Response
: Specify no more then 1 MessageEncodingBindingElement in the binding configuration.

WCFCH0203E
: The endpoint URI address for the service listener must be used exactly as provided.

Explanation
: The binding information for the endpoint URI address must specify a value of 'Explicit' for the 'listenUriMode' parameter.

Response
: Change the parameter value to 'Explicit'.

WCFCH0204E
: SSL is not supported for managed client connections [endpoint URI: '{0}'].

Explanation
: The endpoint URI specifies an SSL connection type which is only supported for unmanaged client connections.

Response
: Modify the channels binding properties to specify an unmanaged client connection mode.

WCFCH0301E-0400E: Channel messages

Use the following information to understand WCFCH0301E-0400E channel messages.

WCFCH0301E
: The URI scheme '{0}' is not supported.

Explanation
: The requested endpoint contains a URI scheme which is not supported by the channel.

Response
: Specify a valid scheme for the channel.

WCFCH0302E
: The received message '{0}' was not a JMS bytes or a JMS text message.

Explanation
: A message has been received but it is not of the correct type. It must be either a JMS bytes message or a JMS text message.

Response
: Check the origin and contents of the message and determine the cause for it being incorrect.

WCFCH0303E
: 'ReplyTo' destination missing.

Explanation
: A reply cannot be sent because the original request does not contain a 'ReplyTo' destination.

Response
: Investigate the reason for the missing destination value.

WCFCH0304E
: The connection attempt to queue manager '{0}' failed for endpoint '{1}'

Explanation
: The queue manager could not be contacted at the given address.

Response
: Review the linked exception for further details.

WCFCH0305E
: The connection attempt to the default queue manager failed for endpoint '{0}'

Explanation
: The queue manager could not be contacted at the given address.

Response
: Review the linked exception for further details.

WCFCH0306E
: An error occurred while attempting to receive data from endpoint '{0}'

Explanation
: The operation could not be completed.

Response
: Review the linked exception for further details.

WCFCH0307E
: An error occurred while attempting to send data for endpoint '{0}'

Explanation
: The operation could not be completed.

Response
: Review the linked exception for further details.

WCFCH0308E
An error occurred while attempting to close the channel for endpoint '{0}'

Explanation
The operation could not be completed.

Response
Review the linked exception for further details.

WCFCH0309E
An error occurred while attempting to open the channel for endpoint '{0}'

Explanation
The operation could not be completed.

Response
The endpoint might be down, unavailable, or unreachable, review the linked exception for further details.

WCFCH0310E
The timeout '{0}' was exceeded while attempting to receive data from endpoint '{0}'

Explanation
The operation did not complete in the time allowed.

Response
Review the system status and configuration and increase the timeout if required.

WCFCH0311E
The timeout '{0}' was exceeded while attempting to send data for endpoint '{0}'

Explanation
The operation did not complete in the time allowed.

Response
Review the system status and configuration and increase the timeout if required.

WCFCH0312E
The timeout '{0}' was exceeded while attempting to close the channel for endpoint '{0}'

Explanation
The operation did not complete in the time allowed.

Response
Review the system status and configuration and increase the timeout if required.

WCFCH0313E
The timeout '{0}' was exceeded while attempting to open the channel for endpoint '{0}'

Explanation
The operation did not complete in the time allowed.

Response
The endpoint might be down, unavailable, or unreachable, review the system status and configuration and increase the timeout if required.

WCFCH0401E-0500E: Binding messages

Use the following information to understand WCFCH0401E-0500E binding messages.

WCFCH0401E
: No context.

Explanation
: An internal error has occurred.

Response
: Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM Support Portal for IBM MQ (see http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ), or the IBM Support Assistant (at http://www.ibm.com/software/support/isa/), to see if a solution is already available. If you cannot find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

WCFCH0402E
: Channel type '{0}' is not supported.

Explanation
: The users application or the WCF service contract has requested a channel shape which is not supported.

Response
: Identify and use a channel shape which is supported by the channel.

WCFCH0403E
: No exporter.

Explanation
: An internal error has occurred.

Response
: Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM Support Portal for IBM MQ (see http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ), or the IBM Support Assistant (at http://www.ibm.com/software/support/isa/), to see if a solution is already available. If you cannot find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

WCFCH0404E
: The WS-Addressing version '{0}' is not supported.

Explanation
: The addressing version specified is not supported.

Response
: Specify an addressing version which is supported.

WCFCH0405E
: No importer.

Explanation
: An internal error has occurred.

Response
: Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM Support Portal for IBM MQ (see http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ), or the IBM Support Assistant (at http://www.ibm.com/software/support/isa/), to see if a solution is already available. If you cannot find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

WCFCH0406E
Endpoint 'Binding' value missing.

Explanation
An internal error has occurred.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM Support Portal for IBM MQ (see http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ), or the IBM Support Assistant (at http://www.ibm.com/software/support/isa/), to see if a solution is already available. If you cannot find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

WCFCH0501E-0600E: Binding properties messages

Use the following information to understand WCFCH0501E-0600E binding properties messages.

WCFCH0501E
The binding property '{0}' has an invalid value '{1}'.

Explanation
An invalid value has been specified for a binding property.

Response
Specify a valid value for the property.

WCFCH0601E-0700E: Async operations messages

Use the following information to understand WCFCH0601E-0700E async operations messages.

WCFCH0601E
The async result parameter '{0}' object is not valid for this call.

Explanation
An invalid async result object has been provided.

Response
Specify a valid value for the parameter.

IBM Support Assistant (ISA)

The IBM Support Assistant (ISA) helps you to resolve questions and problems with IBM software products by providing access to support-related information and troubleshooting tools.

ISA is available at no charge to install on your computer; you then install the relevant product add-ons. ISA has a built-in user guide, and the ISA download package includes a quick start installation and configuration guide. This topic contains a brief overview of the features of the ISA Workbench Version 4; you can find more detailed information on the IBM SupportAssistant web page.

From the ISA home page you can search for information, analyze problems, and collect data to send to IBM.

Find information
Click **Find Information** to search multiple information sources concurrently. You can search the following sources.
- IBM software support documents
- IBM developerWorks®
- IBM news groups and forums
- Google Web search
- Guided troubleshooter content
- Product documentation

Analyze problem
Click **Analyze Problem** to access diagnostic tools or a guided troubleshooter, or to collect data. More tools might be made available periodically, so check for updates by clicking **Find new add-ons**.

Collect and send data
Click **Collect and Send Data** to complete the following tasks.
- Collect diagnostic data automatically from a local or remote computer.
- Send files to IBM Support for problem determination.
- Create and submit a new problem report.
- View or update an existing problem report.

For information on installing the ISA, see "Installing the IBM Support Assistant (ISA)."

Related concepts:

"Troubleshooting overview" on page 1276
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

Related tasks:

"Searching knowledge bases" on page 1638
If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Installing the IBM Support Assistant (ISA)

You can install the IBM Support Assistant (ISA) from the ISA downloads Web page.

Before you begin

Before you start:

Read the concept topic about the "IBM Support Assistant (ISA)" on page 1635.

About this task

Follow these steps to install ISA on your computer:

Procedure

1. Go to the IBM SupportAssistant web page to download the installation package.
2. Log in by using your IBM ID and password. If you do not have an IBM ID, click **Get an IBM ID** to create one.
3. Select the version of ISA that you want and click **Continue**.
4. Click **View license** to read the license agreement in a separate window, then select **I agree** and click **I confirm**.
5. Select the relevant operating system, click **Download now**, and save the compressed file to a temporary directory.
6. Extract the files from the compressed file to a temporary directory. The files that you extract include a quick start guide that tells you how to install, upgrade, and configure ISA.
7. Follow the instructions in the quick start guide to install ISA.

Results

When you have installed ISA successfully, you can use the desktop icon to open it, or you can click **Programs** > **IBM Support Assistant** > **IBM Support Assistant**. (The exact name of the entry in the Start menu depends on the version of ISA that you have installed.)

What to do next

Now that you have installed ISA, install the WMQ add-on, as described in "Updating the IBM Support Assistant (ISA)."

Updating the IBM Support Assistant (ISA)

You can update ISA by installing product and tool add-ons.

Before you begin

Before you start:

1. Read the concept topic about the "IBM Support Assistant (ISA)" on page 1635.
2. Install the IBM Support Assistant.

About this task

To install product add-ons and tool add-ons, complete the following steps.

Procedure

1. Open the IBM Support Assistant by clicking **Programs** > **IBM Support Assistant** > **IBM Support Assistant**.
2. Click **Update** > **Find New** and select either **Product Add-ons** or **Tools Add-ons**.
3. Select the appropriate product add-ons to install and click **Next**. Add-ons are categorized by product family, therefore expand **WebSphere** and select IBM MQ.
4. Select the appropriate tool add-ons and click **Next**.
5. Read and accept the license agreement and click **Next**.
6. Click **Finish** to install the selected add-ons.
7. When the installation completes, click **Finish**, then click **Yes** to restart ISA.

What to do next

When you have installed the add-ons in successfully, click **Find Information** to search various forms of information for the products that you have selected. You can also use ISA to analyze problems and collect and send data to IBM.

Searching knowledge bases

If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

Before you begin

1. If you have not already installed the IBM Support Assistant, you can find instructions about how to do so in "Updating the IBM Support Assistant (ISA)" on page 1637.
2. Install the IBM Support Assistant WMQ plug-in by following the instructions in "Installing the IBM Support Assistant (ISA)" on page 1636.

Procedure

1. **Search the product documentation**

   IBM provides extensive documentation in the form of online product documentation hosted in IBM Knowledge Center. A downloadable version of the product documentation can be installed on your local machine or on a local intranet. You can use the search function of the product documentation to query conceptual and reference information as well as detailed instructions for completing tasks.
2. **Search the IBM database for similar problems**

   IBM keeps records of all known problems with its licensed programs on its software support database (RETAIN). IBM support center staff continually update this database as new problems are found, and they regularly search the database to see if problems they are told about are already known. You can use one of the IBM search tools to search the database, or you can contact IBM support center to perform the search for you. For more information about searching the IBM database, see "Searching the IBM database for similar problems, and solutions" on page 1639.
3. **Search the Internet**

   If you cannot find an answer to your question in the product documentation, search the Internet for the latest, most complete information that might help you resolve your problem, including:
   - IBM technotes
   - IBM downloads
   - IBM Redbooks
   - IBM developerWorks
   - Forums and newsgroups
   - Internet search engines

   You can use the IBM Support Assistant (ISA) to help in your search of knowledge bases. With ISA, you can:
   - Query multiple sources of support information
   - Access available diagnostic tools
   - Collect diagnostic data automatically
   - Send files to IBM Support for problem determination
   - Create and submit a new problem report
   - View or update an existing problem report

   For more information, see "IBM Support Assistant (ISA)" on page 1635.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"IBM Support Assistant (ISA)" on page 1635
The IBM Support Assistant (ISA) helps you to resolve questions and problems with IBM software products by providing access to support-related information and troubleshooting tools.

Related tasks:

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Searching the IBM database for similar problems, and solutions

IBM maintain a database of known problems, and solutions to some of those problems. Use this topic to understand how to best search the database.

IBM keeps records of all known problems with its licensed programs on its software support database (RETAIN). IBM support center staff continually update this database as new problems are found, and they regularly search the database to see if problems they are told about are already known.

If you have access to one of the IBM search tools such as INFORMATION/ACCESS OR INFORMATION/SYSTEM you can look on the RETAIN database yourself. If not, you can contact the IBM support center to perform the search for you.

You can search the database using a string of keywords to see if a similar problem already exists. This section explains how to search the database using keywords.

You can use the keyword string (also called the symptom string) that appears in a dump or SYS1.LOGREC record to search the database, or you can build your own keyword string z/OS from the procedure described in "Building a keyword string" on page 1643. Before you use the procedures in this section, make some initial checks by searching through the following appropriate product documentation section specific to your platform:

- Windows UNIX Linux "Making initial checks on Windows, UNIX and Linux systems" on page 1277
- IBM i "Making initial checks on IBM i" on page 1288
- z/OS "Making initial checks on z/OS" on page 1297

If the search is successful, you find a similar problem description and, usually, a fix. If the search is unsuccessful, you should use these keywords when contacting IBM for additional assistance, or when documenting a possible *authorized program analysis report* (APAR).

Searching the IBM software support database is most effective if you:

- Always spell keywords the way they are spelled in this documentation
- Include all the appropriate keywords in any discussion with your IBM support center

Use the following topics to find out more about searching the IBM database for problems:

- "The search argument process" on page 1640
- "The keyword format" on page 1641
- z/OS "Building a keyword string" on page 1643
- "SDB format symptom-to-keyword cross reference" on page 1655
- "IBM MQ component and resource manager identifiers" on page 1657

The search argument process

Use this topic to understand how to search the RETAIN database.

Use the following procedure when searching the IBM software support database:

1. Using INFORMATION/ACCESS or INFORMATION/SYSTEM, search the database using the keywords you have developed. z/OS Details on how to construct suitable keywords are given in "Building a keyword string" on page 1643

   Note: Do *not* use both the CSECT keyword and the load module modifier keyword at the same time for the first search. z/OS Refer to "Load module modifier keyword" on page 1649 for additional information.
2. Compare each matching APAR closing description with the current failure symptoms.
3. If you find an appropriate APAR, apply the correction or PTF.
4. If you do not find an appropriate APAR, vary the search argument by following the suggestions provided under "Techniques for varying the search process."

Related concepts:

"APARs and PTFs" on page 1664
After a problem is confirmed an APAR can be raised and a PTF released. Use this topic to understand the APAR and PTF process.

Techniques for varying the search process:

You can widen or narrow the scope of your search or you can alter the keywords to make the search more precise.

To vary your search, follow these guidelines:

Dropping keywords to widen your search

If you used a complete set of keywords z/OS (as described in "Building a keyword string" on page 1643) and could not find any problem descriptions to examine, drop one or more of the following keywords and try again:

- Release-level keyword
- Load Module modifier keyword
- Recovery routine modifier keyword
- CSECT keyword

Adding keywords to narrow your search

If you tried to search with an incomplete set of keywords and found too many problem descriptions to examine, add keywords to narrow your search. For example, for storage manager abends (which produce a reason code beginning with X'00E2'), you use the CSECT name recorded in the VRA to narrow or vary the search.

Making your set of keywords more precise

If you tried to search with a complete set of keywords and found too many matching descriptions and if you received a 4-byte IBM MQ abend reason code, you might be able to make your set of keywords more precise. z/OS Look up the 4-byte abend reason code in IBM MQ for z/OS messages, completion, and reason codes to find additional information available for this problem.

Replacing keywords to locate problems

If your type-of-failure keyword is WAIT, LOOP, or PERFM, and if you did not find a matching problem description, replace that keyword with one of the other two listed here. Sometimes a problem that appears to be a performance problem might actually be a WAIT or LOOP; likewise, a problem that seems to be a WAIT or a LOOP might actually be recorded as a performance problem.

Using message numbers in your search
If your type-of-failure keyword is MSGx and you received more than one message near the time of the problem, repeat the search replacing the message number in the keyword with the number of each related message in turn.

Using DOC as a keyword in your search
If your type-of-failure keyword is MSGx, PERFM, or INCORROUT, and if the problem occurred immediately after you performed some action that an IBM MQ documentation told you to perform, the problem could be recorded as a DOC type of failure. In this case, try searching with DOC as your type-of-failure keyword, rather than with MSGx, PERFM, or INCORROUT.

The keyword format

Searches can be performed using free format keywords or structured database (SDB) format keywords. Use this topic to understand how to perform searches using different keyword formats.

The keyword formats

The keywords z/OS in "Building a keyword string" on page 1643 are described in two distinct formats: the z/OS, or free format; and the structured database (SDB) format. Structured symptoms are also called RETAIN symptoms and "failure keywords⌂.

If your installation has a tool for performing structured searches, you can use the SDB format. Otherwise, you should use the free format. For both formats, your choice of keywords depends on the type of failure that occurred.

- Free format
- Structured database (SDB) format

Free format

A free form keyword can consist of any piece of data that is related to the problem. To help you search the database, a set of keywords has been defined, and you can use them to narrow your search. (For example, if you know the name of the CSECT in error, you can use this to search, but if you add the MSGxx or ABEND keyword, your search will be more precise.)

The following list shows keywords defined for use in a free format search:

Table 148. Keywords defined for use in a free format search

Keyword	Meaning
ABEND	Abnormal termination of a task; no error message.
ABENDxx	Abnormal termination of a task; xx is the abend code.
ABENDUxx	User abend; xx is the abend code.
DOC	Documentation discrepancy that caused a problem.
HALTxx	Halt; xx is the halt number.
INCORROUT	Any incorrect data output, except performance degradation.
INTEG	Integrity problem.
LOOP	Loop.
MSGxx	Any message; xx is the message identifier.
PERFM	Performance degradation.

Table 148. Keywords defined for use in a free format search (continued)

Keyword	Meaning
PROCCHK	Processor check.
PROGCH	Program check.
WAIT	Wait condition; undocumented and no identifier.
WAITxx	System wait condition; xx is the identifier.

Structured database (SDB) format

The structured symptoms consist of a prefix keyword, which identifies the type of symptom, followed by a slash (/) and the data portion of the symptom.

- The prefix keyword has one through eight characters.
- All characters must be alphanumeric, #, @, or $.
- At least one character of data is required.
- The maximum length, including the prefix, is 15 characters.

For example, the following is a structured symptom string for a message identifier of CSQC223D:

```
MS/CSQC223D
```

The following list shows the structured symptom strings:

Table 149. Keywords defined for use in a structured format search

Keyword	Meaning
AB	Abend code.
FLDS	Name of a field or control block involved with the problem.
LVLS	Level of the base system or licensed program.
MS	Message identifier.
OPCS	Operation code (opcode) for software, such as an assembler-language opcode.
PCSS	Program command or other software statement, such as JCL, a parameter, or a data set name.
PIDS	Program identifier for a component involved in the problem.
PRCS	Program return code, generated by software, including reason codes and condition codes.
PTFS	Program temporary fix (PTF) for software associated with a problem.
PUBS	Identifier of a publication associated with a problem.
RECS	Record associated with a problem.
REGS	Register for a software program associated with a problem. The value can be the register/PSW difference (*rrddd*), which the STATUS FAILDATA subcommand of IPCS provides for abends. The difference (*ddd*) is a hexadecimal offset from a probable base register or branch register (*rr*).
RIDS	Routine identifier, such as the name of a CSECT or subroutine. If the RIDS/ value has no suffix, the value is a CSECT name. The following suffixes are supported: • #L for a load module • #R for a recovery routine

Table 149. Keywords defined for use in a structured format search (continued)

Keyword	Meaning
VALU	Value in a field or register. One of the following qualifiers is required as the first character of the value: • B for a bit value • C for a character value • H for a hexadecimal value
WS	Wait state code issued by the system, or device-issued wait code. One of the following qualifiers is required as the first character of the value: • D for disabled wait (system disabled for I/O or external interrupts) • E for enabled wait

For more information about which prefix keyword to use for which type of symptom, see "SDB format symptom-to-keyword cross reference" on page 1655.

Building a keyword string

This section describes a systematic way of selecting *keywords* to describe a failure in IBM MQ for z/OS. Keywords are predefined words or abbreviations that identify aspects of a program failure.

To determine which IBM MQ for z/OS keywords to use and the procedures for selecting them, see the flowchart in Figure 114 on page 1644.

To begin selecting your keywords:

1. Follow the procedures in "The component-identifier keyword" on page 1645 and "The release-level keyword" on page 1646. Do this for all failures.
2. Follow one of the type-of-failure keyword procedures.
3. Identify the area of the failure using CSECT and modifier keywords when appropriate. The procedures in this section refer you to these steps as needed.
4. Follow "Searching the IBM database for similar problems, and solutions" on page 1639 to learn how to search the database with your set of keywords. Do this for all failures.

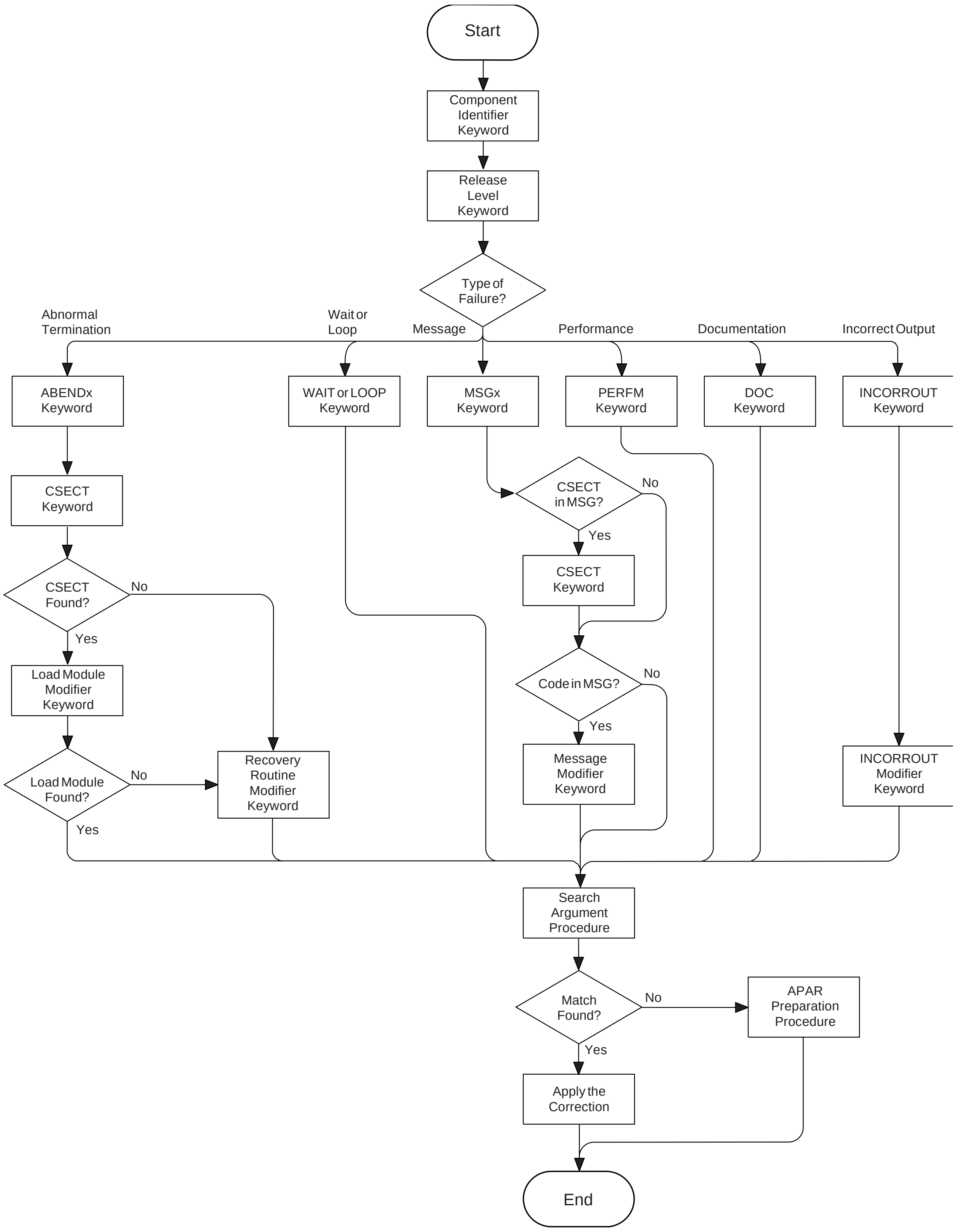


Figure 114. High-level flowchart of various sets of keywords

Use the following topics to build a keyword string for searching the IBM database for problems:

- "The component-identifier keyword" on page 1645
- "The release-level keyword" on page 1646

- "Type-of-failure keyword" on page 1646
- "The abend keyword, and its associated keywords" on page 1647
- "Wait and loop keywords" on page 1650
- "The message keyword" on page 1651
- "Performance keyword" on page 1653
- "Documentation keyword" on page 1653
- "Incorrect output keyword" on page 1654

Related concepts:

"The search argument process" on page 1640
Use this topic to understand how to search the RETAIN database.

"The keyword format" on page 1641
Searches can be performed using free format keywords or structured database (SDB) format keywords. Use this topic to understand how to perform searches using different keyword formats.

"APARs and PTFs" on page 1664
After a problem is confirmed an APAR can be raised and a PTF released. Use this topic to understand the APAR and PTF process.

Related reference:

"SDB format symptom-to-keyword cross reference" on page 1655
You can use structured database (SDB) formats to search the RETAIN database.

"IBM MQ component and resource manager identifiers" on page 1657
Use this topic as a reference for component-identifiers, and resource manager identifiers.

The component-identifier keyword:

You can use the component-identifier as a keyword to search the IBM documentation and software support database for known problems and solutions.

The *component-identifier keyword* identifies the library within the IBM software support database that contains *authorized program analysis reports* (APARs) and *program temporary fixes* (PTFs) for the product.

The component-identifier keyword for IBM MQ for z/OS is **5655R3600** .

This section describes how to determine the nine-digit component identifier keyword for your failure to verify that the problem was caused by IBM MQ for z/OS. If the component identifier is not 5655R3600, the problem could be caused by another product.

If the problem caused a dump, display the dump title, locate the COMP= label, and note the first five characters following that label. If these characters are **R3600** , the problem was caused by IBM MQ for z/OS. Append those five characters to **5655** and use this as the first keyword in your search argument.

```
ssnm,ABN=compltn-reason,U=userid,C=compid.release.comp-function,
M=module, LOC=loadmod.csect+csect_offset
```

If you cannot use the dump title, display the z/OS SYMPTOM STRING in the formatted dump. Note the nine characters following the PIDS/ label.

Related concepts:

"IBM MQ for z/OS dumps" on page 1735
Use this topic for information about the use of dumps in problem determination. It describes the steps you should take when looking at a dump produced by an IBM MQ for z/OS address space.

The release-level keyword:

You can use the release-level as a keyword to search the IBM documentation and software support database for known problems and solutions.

The *release-level keyword* narrows the symptom search to your specific release level. Using this keyword is optional, but suggested, when searching the IBM software support database. It is required, however, when an APAR is submitted.

Locate the three-digit release identifier in the dump title. It follows COMP= R3600, for example:

```
COMP= R3600. 710
```

Add the release-level to your keyword string, in one of the formats shown:

Free format
```
5655R3600  R710
```

Structured format
```
PIDS/ 5655R3600 LVLS/  710
```

Type-of-failure keyword:

You can use the type-of-failure as a keyword to search the IBM documentation and software support database for known problems and solutions.

To narrow your search, use one or more of the type-of-failure and modifier keywords to describe an external symptom of a program failure. The various types of failures are shown in Table 150. Use this table to find the name and page number of the keyword that best matches your problem.

Table 150. Types of IBM MQ for z/OS failures

Problem	Procedure
Abend of the subsystem or task	"The abend keyword, and its associated keywords" on page 1647
Unexpected program suspension	"Wait and loop keywords" on page 1650
Uncontrolled program looping (often signaled by repeating messages or output)	"Wait and loop keywords" on page 1650
Errors signaled by or associated with messages	"The message keyword" on page 1651
Performance degradation	"Performance keyword" on page 1653
Documentation problem	"Documentation keyword" on page 1653
Unexpected or missing output	"Incorrect output keyword" on page 1654

The abend keyword, and its associated keywords:

The abend keyword is often seen in association with other keywords. These keywords can be used together to form a keyword string.

Use the ABEND keyword when the subsystem or task terminates abnormally. This procedure describes how to locate the abend completion code and the abend reason code (if there is one), and how to use them in a set of keywords. Check the SYS1.LOGREC to determine how many abends there were. Sometimes an earlier abend causes a secondary abend that causes a dump. If no dump has been taken, try searching the database with a minimum symptom string (the component-identifier, and release-level keywords). If you cannot find any information that seems to relate to your problem, contact your IBM support center.

When an IBM MQ for z/OS abend occurs, you will see one of the following symptoms:
- An IEA911E message from z/OS, indicating that an SVC dump occurred. See "IEA911E message."
- The CSQV086E message QUEUE MANAGER ABNORMAL TERMINATION REASON=xxxxxxxx. See "CSQV086E message" on page 1648.
- "CSECT keyword" on page 1649
- "Load module modifier keyword" on page 1649
- "Recovery routine modifier keyword" on page 1650

IEA911E message:

The message code IEA911E can be analyzed to give further information about abends.

You can use the information from the IEA911E message to extract further details about the abend code. Use the following steps to assist with the analysis:
1. Use the `DISPLAY DUMP,TITLE` command on the console to display the SVC dump title for this abend, or use one of the methods described in "IBM MQ for z/OS dumps" on page 1735 to look at the dump title in the dump.

   Note: If the first five characters of the COMP field are not R3600, or the dump title is not of the same form as Figure 135 on page 1751 or Figure 136 on page 1752, the problem was not caused by IBM MQ for z/OS, or you are looking at the wrong dump.
2. Locate the 3-character completion code following the word ABND.
   - If the completion code is X'071', or X'122', the operator pressed the RESTART key or canceled the job, probably to break a loop. Verify that this is the case, and turn to "Wait and loop keywords" on page 1650.
   - Otherwise, add this to your keyword string, in one of the formats shown in this topic (in this example, X'0C4' is used):

     Free format
     ```
     5655R3600  R710 ABEND0C4
     ```

     Structured format
     ```
     PIDS/ 5655R3600 LVLS/ 710 AB/S00C4
     ```
3. Some abends also have reason codes. These reason codes are usually found in message CSQV086E, and register 15 at the time of the abend. Locate the reason code for the abend either:
   - In the 4-byte reason code field in a dump title generated by IBM MQ for z/OS
   - In the registers at time of error in the abstract information section of the dump
   - From the value of register 15 in the error summary display
4. If the completion code is X'5C6', review the diagnostic information for the reason code in IBM MQ for z/OS messages, completion, and reason codes. Follow any procedures recommended there.

If the completion code is anything else, and you have found a reason code, check the value against the description of the abend code in the *MVS System Codes* manual to see if it is valid for the abend completion code.

5. Add the reason code to the keyword string (in this example X'00E20015' is used):

   Free format
   ```
   5655R3600  R710 ABEND5C6 RC00E20015
   ```
   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ 710 AB/S05C6 PRCS/00E20015
   ```
   Refer to “CSECT keyword” on page 1649.

CSQV086E message:

The text from the message code CSQV086E can be analyzed to give further information about abends.

You can use the information from the CSQV086E message to extract further details about the abend code. Use the following steps to assist with the analysis:

1. Issue the DISPLAY DUMP command to see whether any SVC dumps occurred near the time the message appeared. (See the *MVS System Commands* manual if necessary.)
2. If there was only one SVC dump for the abend, follow the procedure starting at step 1 on page 1647.
3. If there were two or more SVC dumps, follow the steps here.
   a. Read the sections under IBM MQ for z/OS messages, completion, and reason codes that describe the reason code appearing in your message, and any reason codes appearing in the SVC dump titles. Reason codes appear after the completion code in the SVC dump title. For an example, see “Analyzing the dump and interpreting dump titles” on page 1750.
   b. Compare the reason codes in the SVC dumps to determine which dump relates to the CSQV086E message.
   c. Use that SVC dump and follow the procedure starting at step 1 on page 1647.
4. If there were two or more different abends, follow the steps here:
   a. Determine which abend was the original cause by reviewing the time stamps in the SYS1.LOGREC entries.
   b. Use that SVC dump and follow the procedure starting at step 1 on page 1647.
5. If there were no SVC dumps for the abend, follow the steps here.
   a. Locate the 4-byte reason code in the message.
   b. Review the diagnostic information in IBM MQ for z/OS messages, completion, and reason codes. Follow any procedures recommended there.
   c. Add this to your keyword string, in one of the formats shown in this topic (in this example, a reason code of X'00D93001' is used):

      Free format
      ```
      5655R3600  R710 ABEND6C6 RC00D93001
      ```
      Structured format
      ```
      PIDS/ 5655R3600 LVLS/ 710 AB/S06C6 PRCS/00D93001
      ```
      Refer to “CSECT keyword” on page 1649.

CSECT keyword:

The CSECT keyword and parameter is often associated with an abend code and can assist with identifying where the abend problem occurred.

To find the name of the failing CSECT, locate the LOC= label; the second word following it is the CSECT name. For an example, see "Analyzing the dump and interpreting dump titles" on page 1750.

Any CSECT name you locate should begin with the letters CSQ or CMQ. If you find a CSECT name with a different prefix, the problem is probably not in IBM MQ for z/OS.

Add the CSECT name to your keyword string:

Free format
`5655R3600  R710 ABEND0C4` **`CSQVATRM`**

Structured format
`PIDS/ 5655R3600 LVLS/ 710 AB/S00C4` **`RIDS/CSQVATRM`**

If required, narrow your search further by referring to "Load module modifier keyword."

If you cannot find the CSECT, see "Recovery routine modifier keyword" on page 1650.

Load module modifier keyword:

The Load module modifier keyword and parameter is often associated with an abend code and can assist with identifying where the abend problem occurred.

Use the load module modifier keyword to identify the name of the load module involved if your search using the CSECT keyword was unsuccessful, or yielded too many possible matches:
- If your search was unsuccessful, replace the CSECT name with the load module name and try again.
- If your search yielded too many possible matches, add the load module name to your string to further narrow the search.

All IBM MQ for z/OS load module names begin with CSQ or CMQ. If you follow these instructions and find a load module name with a different prefix, the problem is in another product.

To locate the load module name, locate the first word following the label LOC=. This is the load module name, and it precedes the CSECT name. (For an example, see "Analyzing the dump and interpreting dump titles" on page 1750.)

Add the load module name to your keyword string, or substitute it for the CSECT name as appropriate. If you are using the structured format, follow the name of the module with the characters #L to indicate that this is a load module. Search the database again using the revised keyword string. (See "Searching the IBM database for similar problems, and solutions" on page 1639.)

Free format
`5655R3600  R710 ABEND5C6 RC00E50013` **`CSQSLD1`** `CSQSVSTK` (with load module name and then CSECT name)

`5655R3600  R710 ABEND5C6 RC00E50013` **`CSQSLD1`** (with load module name only)

Structured format
`PIDS/ 5655R3600 LVLS/ 710 AB/S05C6 PRCS/00E50013` **`RIDS/CSQSLD1#L`** `RIDS/CSQSVSTK` (with load module name and then CSECT name)

`PIDS/ 5655R3600 LVLS/ 710 AB/S05C6 PRCS/00E50013` **`RIDS/CSQSLD1#L`** (with load module name only)

Recovery routine modifier keyword:

The Recovery routine modifier keyword and parameter is often associated with an abend code and can assist with identifying where the abend problem occurred.

Include the name of the recovery routine only when you could not determine the names of the CSECT and load module involved at the time of failure, after looking in both the SVC dump and the SYS1.LOGREC entry.

To obtain the recovery routine name, locate the area of the dump title containing the symbol M=. The word following this identifies the functional recovery routine (FRR) or the extended specify task abnormal exit (ESTAE). For an example, see "Analyzing the dump and interpreting dump titles" on page 1750.

Add this word to your keyword string. If you are using the structured format, follow the name of the module with the characters #R to indicate that this is a recovery routine. Search the database (see "Searching the IBM database for similar problems, and solutions" on page 1639).

Free format

```
5655R3600  R710 ABEND5C6 RC00E20015 CSQTFRCV
```

Structured format

```
PIDS/ 5655R3600 LVLS/ 710 AB/S05C6 PRCS/00E20015 RIDS/CSQTFRCV#R
```

Wait and loop keywords:

The keywords wait and loop can be used as part of a string for searching the IBM documentation and IBM software support database. This can assist in identifying known problems and resolutions.

If the problem occurred immediately after you did something an IBM MQ manual told you to do, the problem might be related to the documentation. If you think that this is the case, see "Documentation keyword" on page 1653.

If you have verified that the wait or loop problem cannot be resolved through other means, use the following procedure:

1. Add WAIT or LOOP to your keyword string, in one of the formats shown in this topic (in this example **WAIT** is used).

   Free format

   ```
   5655R3600  R710 WAIT
   ```

   Structured format

   ```
   PIDS/ 5655R3600 LVLS/ 710 WAIT
   ```
2. See "Searching the IBM database for similar problems, and solutions" on page 1639.

The message keyword:

The message keyword can be used to search the IBM documentation and software support database for known problems and solutions

Use the MSG keyword if an error is associated with an IBM MQ for z/OS message. If you received multiple messages for one error, search the database using the first message issued. If unsuccessful, search the database using the next message, then the next, and so on.

To see if other messages related to your problem have been issued, check the console for IBM MQ for z/OS messages, as well as messages issued by other products. If any message is prefixed with "IEC⌂, indicating it was issued by data management services, check the SYSLOG for messages that identify associated data set problems. SYSLOG can also help to diagnose user errors.

If your message was issued immediately after you did something that an IBM MQ manual told you to do, the problem might be related to the documentation rather than to the message. If this is the case, refer to “Documentation keyword” on page 1653. Otherwise, compare the message prefix with those shown in the table Message Prefixes to determine the appropriate procedure to follow.

Table 151. Message prefixes

Prefix	**Component**	**Procedure**
AMQ	IBM MQ (not z/OS)	Consult IBM MQ messages
AMT	IBM MQ	Consult the Business Integration - IBM MQ SupportPacs for details of the MA0F SupportPac
ATB	APPC	Consult *MVS System Messages*
ATR	Resource recovery services	Consult *MVS System Messages*
CBC	C/C++	Consult *C/MVS User's Guide*
CEE	Language Environment	Consult *Language Environment Debugging Guide and Run-Time Messages*
CSQ	IBM MQ for z/OS	Follow “Procedure for IBM MQ for z/OS messages” on page 1652
CSV	Contents supervision	Consult *MVS System Messages*
DFH	CICS	Consult *CICS Messages and Codes*
DFS	IMS	Consult *IMS/ESA Messages and Codes*
DSN	Db2	Consult *Db2 Messages and Codes*
EDC	Language Environment	Consult *Language Environment Debugging Guide and Run-Time Messages*
EZA, EZB, EZY	TCP/IP	Consult *z/OS V2R6.0 eNetwork CS IP Messages: Volumes 1, 2, and 3*
IBM	Language Environment	Consult *Language Environment Debugging Guide and Run-Time Messages*
ICH	RACF	Consult *z/OS Security Server (RACF) Messages and Codes*
IDC	Access method services	Consult *MVS System Messages*
IEA	z/OS system services	Consult *MVS System Messages*
IEC	Data management services	Consult *MVS System Messages*
IEE, IEF	z/OS system services	Consult *MVS System Messages*
IKJ	TSO	Consult *MVS System Messages*
IST	VTAM	Consult *VTAM Messages*
IWM	MVS workload management services	Consult *MVS System Messages*

Table 151. Message prefixes (continued)

Prefix	Component	Procedure
IXC	Cross-system coupling facility (XCF)	Consult *MVS System Messages*
IXL	Cross-system Extended Services (XES)	Consult *MVS System Messages*

Related concepts:

"Procedure for IBM MQ for z/OS messages"
The IBM MQ messages can be analyzed for the possible cause of an error. To do this you must analyze the text of the message and use the components of the text to search the IBM software support database.

Procedure for IBM MQ for z/OS messages:

The IBM MQ messages can be analyzed for the possible cause of an error. To do this you must analyze the text of the message and use the components of the text to search the IBM software support database.

Analyze the text of the any IBM MQ messages and use the parts of the text, for example CSECT names, variables and message codes, to search IBM software support database for known problems and resolutions.

1. Check whether the name of the CSECT issuing the message appears. This name follows the message number. If no CSECT name appears, only one CSECT can issue this message.
2. Determine whether the message contains any variables, such as return or reason codes.
3. If no CSECT name appears, add the message number to your keyword string, in one of the formats shown in this topic (in this example, message CSQJ006I is used):

   Free format
   ```
   5655R3600  R710 MSGCSQJ006I
   ```
   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ R710 MS/CSQJ006I
   ```
4. If a CSECT name does appear, add both the message number and the CSECT name to your keyword string, in one of the formats shown here (in this example, a message number of CSQJ311I and a CSECT name of CSQJC005 are used):

   Free format
   ```
   5655R3600  R710 MSGCSQJ311I CSQJC005
   ```
   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ 710 MS/CSQJ311I RIDS/CSQJC005
   ```
5. If the message contains return or reason codes, add these to your keyword string, in one of the formats shown in this topic:

   Free format
   ```
   5655R3600  R710 MSGCSQM002I RCE
   ```
   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ R710 MS/CSQM002I PRCS/0000000E
   ```
6. If the message contains any other types of variables, append them to your keyword string.

   Free format
   ```
   5655R3600  R710 MSGCSQJ104I OPEN
   ```
   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ R710 MS/CSQJ1041 MS/OPEN
   ```
7. See "Searching the IBM database for similar problems, and solutions" on page 1639.

Performance keyword:

The performance keyword can be used to search the IBM documentation and software support database for known problems and solutions.

You can resolve most performance problems through system tuning, which should be handled by the IBM MQ for z/OS system administrator. Before following the procedure show in this topic, use this checklist to verify that the performance problem cannot be resolved through other means:

- See "Dealing with performance problems on z/OS" on page 1755 to determine if you can change the way you have designed your IBM MQ for z/OS subsystem and applications to improve their performance.
- Verify that the performance problem is not related to a WAIT or LOOP. See "Dealing with applications that are running slowly or have stopped on z/OS" on page 1756.
- If the problem occurred immediately after you did something an IBM MQ manual told you to do, the problem might be related to the manual. See "Documentation keyword."
- If performance degraded after someone tuned IBM MQ for z/OS, verify that the tuning options selected were appropriate. Perhaps you can resolve the problem by choosing other options.

If you have verified that the performance problem cannot be resolved through other means, use the following procedure:

1. Record the actual performance, expected performance, and source of expected performance criteria.
2. Add PERFM to your keyword string, as shown in the following example, and see "Searching the IBM database for similar problems, and solutions" on page 1639.

   Free format
   ```
   5655R3600  R710 PERFM
   ```
   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ 710 PERFM
   ```
3. If required, you can narrow your search by adding free-format keywords that describe what you were doing when you experienced the performance problem.

Documentation keyword:

The documentation keyword can be used to search the IBM software support database for known problems and solutions.

The DOC keyword identifies problems caused by incorrect or missing information in an IBM MQ manual. It is possible that a documentation problem could be detected when trying to resolve problems with messages, incorrect output, and performance.

Use the following procedure if you need to use the DOC keyword in your keyword string:

1. Locate the incomplete or erroneous information. Note the page, or topic number, and describe the error and the resulting problem.
2. Add the document number, hyphens omitted, to your keyword string, in one of the formats shown here (in this example, the document number for this manual (GC34-6600-01) is used):

   Free format
   ```
   5655R3600  R710 DOC GC34660001
   ```
   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ 710 PUBS/ GC34660001
   ```

   See "Searching the IBM database for similar problems, and solutions" on page 1639.

   If your search is unsuccessful, follow Step 3 on page 1654.

3. Broaden your search by replacing the last two digits with two asterisks (**). This searches for all problems on that document, rather than on a specific release of the document.

   Free format
   ```
   5655R3600  R710 DOC GC346600 **
   ```

   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ 710 PUBS/GC346600 **
   ```

   If your search is unsuccessful, follow Step 4.
4. If the problem is severe, consider initiating a DOC APAR. Use the information gathered in Step 1, and see "APARs and PTFs" on page 1664.
5. If the problem is less severe, send your comments electronically by clicking the `Feedback` option at the end of each product documentation topic.

   Corrections resulting from readers' comments are included in future editions of the manual but are not included in the software support database.

Incorrect output keyword:

You can use the incorrect output keywords to search the IBM software support database for known problems and solutions.

Use the INCORROUT keyword when output was expected but not received, or when output was different from expected. However, if this problem occurred after you did something that IBM MQ documentation told you to do, the documentation could be in error. If this is the case, see "Documentation keyword" on page 1653.

1. Add **INCORROUT** to your existing keyword string.

   Free format
   ```
   5655R3600  R710 INCORROUT
   ```

   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ 710 INCORROUT
   ```
2. Determine the function and secondary modifier keywords for your problem from Table 152 and Table 153 on page 1655.
3. Add the modifier keywords to your string and use it to search the database. See "Searching the IBM database for similar problems, and solutions" on page 1639.

   Free format
   ```
   5655R3600  R710 INCORROUT RECOVERY BACKOUT
   ```

   Structured format
   ```
   PIDS/ 5655R3600 LVLS/ 710 INCORROUT RECOVERY BACKOUT
   ```

Table 152. INCORROUT modifier keywords: RECOVERY

Secondary keywords	Problem occurrence
none	During recovery
BACKOUT	At backout time
CHECKPOINT	At checkpoint time
COMMIT	At commit time
LOGGING	During logging
RECOVER	During attempt to recover in-doubt
RESTART	During restart process

Table 153. INCORROUT modifier keywords: UTILITY

Secondary keywords	Problem occurrence
none	While running a utility
CSQ1LOGP	While running CSQ1LOGP
CHANGE LOG	While using the Change Log Inventory utility or CSQJUFMT
PRINT LOGMAP	While using the Print Log Map utility
COMMAND	While running the COMMAND function or SDEFS
COPY	While running the COPY function or SCOPY
EMPTY	While running the EMPTY function
FORMAT	While running the FORMAT function, COPYPAGE, RESETPAGE or PAGEINFO
LOAD	While running the LOAD function
CSQUDLQH	While running the dead.letter.queue handler
CSQ5PQSG	While running CSQ5PQSG

SDB format symptom-to-keyword cross reference

You can use structured database (SDB) formats to search the RETAIN database.

Structured database (SDB) format is one of the formats that you can use for searching the RETAIN database. Structured symptoms are also called RETAIN symptoms and "failure keywords⌂. Table 154 lists which prefix keyword to use for which symptom.

“Searching the IBM database for similar problems, and solutions” on page 1639 provides details about searching RETAIN.

Table 154. SDB format symptom-to-keyword cross-reference

Symptom	Keyword	Symptom	Keyword
abend	AB/	access method	RIDS/
address	ADRS/	APAR	PTFS/
assembler macro	RIDS/	assembler message	MS/
CLIST	RIDS/	command	PCSS/
compiler message	MS/	completion code	PRCS/
component	PIDS/	condition code	PRCS/
control block	FLDS/	control block offset	ADRS/
control register	REGS/	CSECT	RIDS/
data set name	PCSS/	dependent component	PIDS/
device error code	PRCS/	disabled wait (coded)	WS/
displacement	ADRS/	display	DEVS/
document	PUBS/	DSECT	FLDS/
enabled wait (coded)	WS/	error code	PRCS/
EXEC	RIDS/	feedback code	PRCS/
field	FLDS/	field value	VALU/
file mode	PCSS/	file name	PCSS/
file type	PCSS/	flag	FLDS/
floating-point register	REGS/	full-screen mode	PCSS/

Table 154. SDB format symptom-to-keyword cross-reference (continued)

Symptom	Keyword	Symptom	Keyword
function key	PCSS/	general purpose register	REGS/
hang	WS/	hung user or task	WS/
I/O operator codes	OPCS/	incorrect output	INCORROUT*
JCL card	PCSS/	JCL parameter	PCSS/
job step code	PRCS/	key	PCSS/
label, code	FLDS/	language statement	PCSS/
level	LVLS/	library name	PCSS/
line command	PCSS/	loop	LOOP*
low core address	ADRS/	machine check	SIG/
macro as a routine	RIDS/	macro as a statement	PCSS/
maintenance level	PTFS/	message	MS/
module	RIDS/	offset	ADRS/
opcode	OPCS/	operator command	PCSS/
operator key	PCSS/	operator message	MS/
option	PCSS/	overlay	OVS/
PA key	PCSS/	panel	RIDS/
parameter	PCSS/	performance	PERFM*
PF key	PCSS/	procedure name	PCSS/
process name	PCSS/	profile option	PCSS/
program check	AB/	program id	RIDS/
program key	PCSS/	program statement	PCSS/
PSW	FLDS/	PTF, PE or otherwise	PTFS/
publication	PUBS/	PUT level	PTFS/
reason code	PRCS/	register value	VALU/
register	REGS/	release level	LVLS/
reply to message	PCSS/	reply to prompt	PCSS/
request code	OPCS/	response to message	PCSS/
response to prompt	PCSS/	return code	PRCS/
routine	RIDS/	service level	PTFS/
special character	PCSS/	SRL	PUBS/
statement	PCSS/	status code	PRCS/
step code	PRCS/	structure word	FLDS/
subroutines	RIDS/	SVC	OPCS/
SYSGEN parameter	PCSS/	system check	PRCS/
table	FLDS/	terminal key	PCSS/
value	VALU/	variable	FLDS/
wait (coded)	WS/	wait (uncoded)	WAIT*

Note: An asterisk (*) indicates that there is no prefix keyword for this type of problem. Use the type-of-failure keyword shown for searches of the software support database.

IBM MQ component and resource manager identifiers

Use this topic as a reference for component-identifiers, and resource manager identifiers.

The component-identifier keyword identifies the library within the IBM software support database that contains *authorized program analysis reports* (APARs) and *program temporary fixes* (PTFs) for the product. Resource manager identifiers (RMIDs) are used to limit the volume of data collected in a trace.

Table 155. IBM MQ component and resource manager identifiers

ID	Prefix	Hex ID	Component name	RMID
-	AMT	-	AMI	-
-	CMQ	-	Application header files	-
m	CSQm	X'94'	Connection manager	148
t	CSQt	X'A3'	Topic manager	163
A	CSQA	X'C1'	Application interface	-
B	CSQB	X'C2'	Batch adapter	-
C	CSQC	X'C3'	CICS adapter	-
E	CSQE	X'C5'	coupling facility manager	197
F	CSQF	X'C6'	Message generator	24
G	CSQG	X'C7'	Functional recovery manager	199
H	CSQH	X'C8'	Security manager interface	200
I	CSQI	X'C9'	Data manager	201
J	CSQJ	X'D1'	Recovery log manager	4
L	CSQL	X'D3'	Lock manager	211
M	CSQM	X'D4'	Message manager	212
N	CSQN	X'D5'	Command server	213
O	CSQO	X'D6'	Operations and control	-
P	CSQP	X'D7'	Buffer manager	215
Q	CSQQ	X'D8'	IMS adapter	-
R	CSQR	X'D9'	Recovery manager	3
S	CSQS	X'E2'	Storage manager	6
T	CSQT	X'E3'	Timer services	227
U	CSQU	X'E4'	Utilities	-
V	CSQV	X'E5'	Agent services	2
W	CSQW	X'E6'	Instrumentation facilities	16, 26
X	CSQX	X'E7'	Distributed queuing	231
Y	CSQY	X'E8'	Initialization procedures and general services	1
Z	CSQZ	X'E9'	System parameter manager	12
1	CSQ1	X'F1'	Service facilities	-
2	CSQ2	X'F2'	IBM MQ - IMS bridge	242
3	CSQ3	X'F3'	Subsystem support	7, 8
4	CSQ4	X'F4'	Sample programs	-
5	CSQ5	X'F5'	Db2 manager	245
6	CSQ6	X'F6'	Customization	-
7	CSQ7	X'F7'	Dump formatting	-

Table 155. IBM MQ component and resource manager identifiers (continued)

ID	Prefix	Hex ID	Component name	RMID
8	CSQ8	X'F8'	Installation	-
9	CSQ9	X'F9'	Generalized command preprocessor	23
-	IMQ	-	C++ bindings	-

Contacting IBM Software Support

Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Before you begin

Ensure that your company has an active IBM software subscription and support contract, and that you are authorized to submit problems to IBM:

- If you use IBM distributed software products (including, but not limited to, Tivoli, Lotus, and Rational® products, as well as Db2 and WebSphere products that run on Windows or UNIX and Linux operating systems), enroll in Passport Advantage® in one of the following ways:
  - Online: Go to the Passport Advantage web page, then click the **Subscription and Support** tab.
  - By telephone: Go to the Software Support Handbook, click **Contacts**, then **Worldwide contacts**.
- If you use IBM eServer software products, you can purchase a software subscription and support agreement by working directly with an IBM marketing representative or an IBM Business Partner. These eServer products include, but are not limited to, Db2 and WebSphere products that run in zSeries, pSeries, and iSeries environments.
- You might also have an IBMLink, CATIA, Linux, S/390, iSeries, pSeries, zSeries, or other support agreement.
- If you are not sure what type of software subscription and support contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States or, from other countries, go to the Software Support Handbook, click **Contacts**, then **Worldwide contacts**.

About this task

Follow the steps in this topic to fully describe the problem and contact IBM Software Support.

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support might create an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround for you to implement until the APAR is resolved and a fix is delivered.

IBM publishes resolved APARs on the IBM product support web pages daily, so that other users who experience the same problem can benefit from the same resolutions.

Procedure

1. Determine the business severity level for the problem.

   When you report a problem to IBM, you will be asked to supply a severity level. Therefore, you need to understand and assess the effect on your business of the problem that you are reporting. Use the following criteria:

Severity	**Effect on business**
Severity 1	**Critical** effect on business: You are unable to use the program, resulting in a critical effect on operations. This condition requires an immediate solution.
Severity 2	**Significant** effect on business: The program is usable but is severely limited.
Severity 3	**Some** effect on business: The program is usable with less significant features (not critical to operations) unavailable.
Severity 4	**Minimal** effect on business: The problem has little effect on operations, or a reasonable workaround to the problem has been implemented.

When deciding the severity of the problem, take care not to understate it, or to overstate it. The support center procedures depend on the severity level so that the most appropriate use can be made of the center's skills and resources. A severity level 1 problem is normally dealt with immediately.

2. Describe the problem and gather background information.

   You might find the information you need in your own in-house tracking system for problems. You can also use the IBM Support Assistant to collect data automatically, as described in the next step, and you can use a "Problem reporting sheet" on page 1660.

   Be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you to solve the problem efficiently. To save time, know the answers to these questions:

   - What was the source of the problem within your system software; that is, the program that seems to be the cause of the problem.
   - What software versions were you running when the problem occurred?
   - Do you have logs, traces, and messages that are related to the problem symptoms?
   - Can the problem be re-created? If so, what steps led to the failure?
   - Have any changes been made to the system? For example:
     - Hardware changes
     - Operating system upgrades
     - Networking software updates
     - Changes in the level of licensed programs
     - PTFs applied
     - Additional features used
     - Application programs changed
     - Unusual operator action
     - z/OS Regenerations
   - Are you currently using a workaround for this problem? If so, be prepared to explain it when you report the problem.

   z/OS See also "Extra inputs needed when reporting z/OS problems" on page 1661.

3. Report the problem to IBM Software Support.

   You can submit a problem report in one of three ways:

   - Use the IBM Support Assistant. To collect data automatically and submit a request to IBM Software Support, open the IBM Support Assistant and click **Service**. For more information, see "IBM Support Assistant (ISA)" on page 1635.
   - Submit a New service request online.
   - Telephone the support center for your country. To find the number to call, go to the Software Support Handbook, click **Contacts**, then **Worldwide contacts**. Your first contact at the support center is the call receipt operator, who takes initial details and puts your problem on a queue. You are then contacted by a support center representative.

When you contact the support center, whether by telephone or electronically, you need to state the name of your organization and your *access code*. Your access code is a unique code authorizing you to use IBM Software Support, and you provide it every time you contact the center. This information is used to access your customer profile, which contains information about your address, relevant contact names, telephone numbers, and details of the IBM products at your installation.

The support center needs to know whether this is a new problem, or a further communication regarding an existing one. If it is new, it is assigned a unique *incident number*. A *problem management record* (PMR) is opened on the RETAIN system, where all activity associated with your problem is recorded. The problem remains "open" until you are in agreement with the support center that it has been resolved and can now be closed. Make a note of the incident number. The support center expects you to quote the incident number in all future communications connected with this problem.

You might be asked to give values from a formatted dump or trace table, or to carry out some special activity, for example to set a trap, or to use trace with a specific type of selectivity, and then to report the results. You will be given guidance by the support center on how to obtain this information.

What to do next

You can inquire any time at your support center on how your PMR is progressing, particularly if it is a problem of high severity.

How your problem is then progressed depends on its nature. The representative who handles the problem gives you guidance. The possibilities are described in "What happens next" on page 1663.

Related tasks:

"Getting product fixes" on page 1665
A product fix might be available to resolve your problem. You can determine what fixes are available by launching a query from the IBM Support Assistant.

Problem reporting sheet

A problem reporting sheet helps you gather the background information that you need when reporting a problem to the IBM support center.

There are two advantages to using a problem reporting sheet when you contact the IBM support center:

- In a telephone conversation, you are better prepared to respond to questions if you have all your findings before you on a sheet of paper.
- You can use the information for planning, organizing, and establishing your priorities for controlling and resolving the problem.

```
 PROBLEM REPORTING SHEET

 Date                     Severity                  Problem  No.

                                                    Incident No.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 Problem/Inquiry

 Abend                    Incorrout                 z/OS Release

 Wait                     Module                    IBM MQ Release

 Loop                     Message                   CICS Release

 Performance              Other                     IMS Release

                                                    DB2 Release
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 Documentation available

 Abend                    System dump               Program output

 Message                  Transaction dump          Other

 Trace                    Translator output

 Symptom string           Compiler output

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 Actions

 Date          Name              Activity








- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 Resolution

 APAR                     PTF                       Other
```

Figure 115. Sample problem reporting sheet

Extra inputs needed when reporting z/OS problems

z/OS

When you report an IBM MQ for z/OS problem to IBM Support, you need to provide additional inputs to those that are needed for other platforms.

The core documentation that is required for reporting an IBM MQ problem on any platform is described in “Contacting IBM Software Support” on page 1658.

As a rule, the total documentation you need to submit for a problem includes all the material you need yourself to do problem determination. Some of the documentation is common to all IBM MQ for z/OS problems, and some is specific to particular types of problem.

Make sure that the problem you have described can be seen in the documentation you send. If the problem has ambiguous symptoms, you need to reveal the sequence of events leading to the failure. Tracing is valuable in this respect but you need to provide details that trace cannot give. You are encouraged to annotate your documentation, if your annotation is legible and does not cover up vital information. You can highlight any data in hardcopy you send, using transparent highlighting markers. You can also write notes in the margins, preferably using a red pen so that the notes are not overlooked.

Finally, note that if you send too little documentation, or if it is unreadable, the support team will have to return your problem marked "insufficient documentation". It is, therefore, worthwhile preparing your documentation carefully and sending everything relevant to the problem.

The additional documentation needed is described in the next section. However, these are only guidelines. You must find out from the IBM support center representative precisely what documentation you need to send for your specific problem.

Additional documentation needed for problems with IBM MQ for z/OS

Here is a list of the additional documentation you might be asked to submit:

- Any hardcopy or softcopy illustrating the symptoms of the problem.
- The dump of the problem, see “Getting a dump” on page 1736.
- The appropriate SYS1.LOGREC records, see “SYS1.LOGREC information” on page 1754.
- The system log.
- A portion of the IBM MQ for z/OS job log.
- Trace records, see “Using trace for problem determination on z/OS” on page 1697.
- Trace information produced by the CICS or IMS adapter, see “Other types of trace” on page 1703.
- Buffer pool statistics, see the sections on using SMF and type 115 records in Configuring z/OS.
- Listings of relevant application programs.
- A list of PTFs and APARs applied.

  Use the following sample JCL produce a list of all the PTFs, APARs, user modifications, and product code that has been installed in the global zone specified by SMPCSI:

  ```
  //SMPELIST EXEC PGM=GIMSMP,REGION=4096K
  //SMPCSI  DD DSN=shlqual.global.csi,DISP=SHR
  //SYSPRINT DD SYSOUT=*
  //SMPCNTL  DD *
  SET BOUNDARY(GLOBAL) .
  LIST
  SYSMODS
  APARS
  FUNCTIONS
  PTFS
  USERMODS
  ALLZONES .
  /*
  ```
- Dump of coupling facility administration structure, see Figure 132 on page 1738.
- Dump of coupling facility application structure, see Figure 132 on page 1738.
- Dump of other queue managers in queue-sharing group, see Figure 128 on page 1737.
- Definitions of the objects in your system.

  You can find these by using the DISPLAY command for each type of object, for example:

  ```
  DISPLAY QUEUE (*) ALL
  ```

  Or, if you regularly make a backup of your object using the MAKEDEF feature of CSQUTIL COMMAND function, the output from that backup job.
- Your IBM MQ Db2 tables.

You can get these by using the sample job CSQ45STB in thlqual.SCSQPROCS to produce a report of all the Db2 tables used by IBM MQ.

You might also be asked for the IBM MQ for z/OS symptom string, or for any keywords associated with the problem. The primary keywords are ABEND, WAIT, LOOP, PERFM, INCORROUT, MSG, and DOC, corresponding exactly with the problem classification types used in “Building a keyword string” on page 1643. Strings containing other keywords are also useful. These are not predefined, and might include such items as a message or message number, an abend code, any parameters known to be associated with the problem, or, for example, STARTUP or INITIALIZATION. The keywords are then used by IBM Support as search arguments, to see if your problem is a known one.

Because of the size of SVC dumps in the cross memory environment, you might want to transfer the SYS1.DUMPxx data set to a tape or like device. You can use the PRDMP service aid program to transfer the SYS1.DUMPxx data set contents to another data set for archiving until the problem is resolved. Alternatively, your support center representative might give you the address of an FTP site where you can send your dump electronically. Each tape should be sent with the following associated information:

- The PMR number assigned by IBM
- A list of data sets on the tape (application source program, JCL, or data)
- A list of how the tape was made, including:
  - The exact JCL listing or the commands used
  - The recording mode and density
  - Tape labeling
  - The record format, logical record length, and block size used for each data set

When the support team receives the package, this is noted on your PMR record on the RETAIN system.

What happens next

After you contact the IBM support center, details of your problem are passed to the appropriate support group. The problems are dealt with in order of receipt and severity level.

At first, an IBM support center representative uses the keywords that you have provided to search the RETAIN database. If your problem is found to be one already known to IBM, and a fix has been devised for it, a *program temporary fix* (PTF) can be dispatched to you quickly. Alternatively, you might be asked to try running your installation using different settings.

If the RETAIN search is unsuccessful, you might be asked to provide more information about your problem.

If the problem requires a change to the code or documentation, an *authorized program analysis report* (APAR) is submitted. This is dealt with by the IBM support group or *change team* and provides a means of tracking the change.

It might be necessary to have several follow-up communications, depending on the complexity of the symptoms and your system environment. In every case, the actions taken by you and the support center are entered in the original PMR. The representative can then be acquainted with the full history of the problem before the next communication.

APARs and PTFs

After a problem is confirmed an APAR can be raised and a PTF released. Use this topic to understand the APAR and PTF process.

An APAR

An *authorized program analysis report* (APAR) is the means by which a problem with an IBM program is documented, tracked, and corrected. It is also used to track problems with IBM documents.

An APAR is raised by the IBM change team when a new problem is reported for which a program or documentation change is required. It is separate to the PMR that is raised when you report first report the problem.

When the change team solves the problem, they might produce a local fix enabling you to get your system running properly again. Finally, a *program temporary fix* (PTF) is produced to replace the module in error, and the APAR is closed.

The APAR process

The first step in the APAR process is that an IBM support center representative enters your APAR into the RETAIN system. The APAR text contains a description of your problem. If you have found a means of getting round the problem, details of this are entered as well. Your name is also entered, so that the support center knows whom to contact if the change team needs to ask anything further about the APAR documentation.

When the APAR has been entered, you are given an APAR number. You must write this number on all the documentation you submit to the change team. This number is always associated with the APAR and its resolution and, if a code change is required, with the fix as well.

During the APAR process, the change team might ask you to test the fix on your system.

Lastly, you need to apply the PTF resulting from the APAR when it becomes available.

Applying the fix

When the change team have created a fix for your problem, they might want you to test it on your system.

When the team is confident that the fix is satisfactory, the APAR is certified and the APAR is closed.

Occasionally, the solution to the APAR requires a change to the documentation only. In some circumstances, the APAR might be closed with a classification code of **FIN**, which means that if there is a subsequent release of IBM MQ, a fix for this problem can be provided at this time.

The APAR becomes a PTF

If the solution requires a code change to the current release, when the APAR is closed the change is distributed as a PTF.

If you want a PTF to resolve a specific problem, you can order it explicitly by its PTF number through the IBM support center. For more information, see the IBM MQ Support, Migration PTFs web page.

Getting product fixes

A product fix might be available to resolve your problem. You can determine what fixes are available by launching a query from the IBM Support Assistant.

Before you begin

1. If you have not already installed the IBM Support Assistant, you can find instructions about how to do so in "Installing the IBM Support Assistant (ISA)" on page 1636.
2. Install the IBM Support Assistant WMQ plug-in by following the instructions in "Updating the IBM Support Assistant (ISA)" on page 1637.

About this task

To launch a query from the IBM Support Assistant:

1. Open the IBM Support Assistant from the Start menu by clicking **Programs** > **IBM Support Assistant** > **IBM Support Assistant**.
2. Click **Product Information**, **WMQ**, **Support page**, **Download**, then **Recommended fixes**. This Web page provides links to the latest available maintenance for the in-service products.

To receive weekly e-mail notifications about fixes and other news about IBM products, follow these steps.

Procedure

1. From the support site (IBM MQ support web page), click **Subscribe to this product** in the Notifications box on the page.
2. If you have already registered for "Notifications", skip to the next step. If you have not registered, click **register now** on the sign-in page and follow the on-screen instructions.
3. Sign in to "My notifications".
4. Click the **Subscribe** tab. A list of products families is shown.
5. In the Software column, click **WebSphere**. A list of products is shown.
6. Select the product for which you want to receive notifications (for example, **IBM MQ**), then click **Continue**.
7. Set options to determine what notifications you receive, how often you receive them, and to which folder they are saved, then click **Submit**.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 1276
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

Related tasks:

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

"Searching knowledge bases" on page 1638
If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

Related information:

Applying and removing maintenance

First Failure Support Technology (FFST

First Failure Support Technology™ (FFST) for IBM MQ provides information about events that, in the case of an error, can help IBM support personnel to diagnose the problem.

First Failure Data Capture (FFDC) provides an automated snapshot of the system environment when an internal event occurs. In the case of an error, this snapshot is used by IBM support personnel to provide a better understanding of the state of the system and IBM MQ when the problem occurred.

The information about an event is contained in an FFST file. In IBM MQ, FFST files have a file type of FDC. FFST files do not always indicate an error. An FFST might be informational.

Monitoring and housekeeping

Here are some tips to help you with managing FFST events:

- Monitor FFST events for your system, and ensure that appropriate and timely remedial action is taken when an event occurs. In some cases, the FDC files might be expected and can therefore be ignored, for example FFST events that arise when IBM MQ processes are ended by the user. By appropriate monitoring, you can determine which events are expected, and which events are not.
- FFST events are also produced for events outside IBM MQ. For example, if there is a problem with the IO subsystem or network, this problem can be reported in an FDC type file. These types of event are outside the control of IBM MQ and you might need to engage third parties to investigate the root cause.
- Ensure that good housekeeping of FFST files is carried out. The files must be archived and the directory or folder must be cleared to ensure that only the most recent and relevant FDC files are available, should the support team need them.

Use the information in the following links to find out the names, locations, and contents of FFST files in different platforms.

- "FFST: IBM MQ classes for JMS" on page 1667
- Windows "FFST: IBM MQ for Windows" on page 1670
- UNIX Linux "FFST: IBM MQ for UNIX and Linux systems" on page 1672
- IBM i "FFST: IBM MQ for IBM i" on page 1674
- "FFST: IBM MQ for HP Integrity NonStop Server" on page 1676

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 1276
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

"Using logs" on page 1678
There are a variety of logs that you can use to help with problem determination and troubleshooting.

z/OS "Problem determination on z/OS" on page 1724
IBM MQ for z/OS, CICS, Db2, and IMS produce diagnostic information which can be used for problem determination.

Related tasks:

"Using trace" on page 1686
You can use different types of trace to help you with problem determination and troubleshooting.

"Searching knowledge bases" on page 1638
If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

FFST: IBM MQ classes for JMS

Describes the name, location, and contents of the First Failure Support Technology (FFST) files that are generated by the IBM MQ classes for JMS.

When using the IBM MQ classes for JMS, FFST information is recorded in a file in a directory that is called `FFDC`, which by default is a subdirectory of the current working directory for the IBM MQ classes for JMS application that was running when the FFST was generated. If the property com.ibm.msg.client.commonservices.trace.outputName has been set in the IBM MQ classes for JMS configuration file, the FFDC directory is a subdirectory of the directory that the property points to. For information about the IBM MQ classes for JMS , see The IBM MQ classes for JMS configuration file.

An FFST file contains one FFST record. Each FFST record contains information about an error that is normally severe, and possibly unrecoverable. These records typically indicate either a configuration problem with the system or an internal error within the IBM MQ classes for JMS .

FFST files are named `JMSC` *`nnnn`*`.FDC`, where *nnnn* starts at 1. If the full file name already exists, this value is incremented by one until a unique FFST file name is found.

An instance of an IBM MQ classes for JMS application writes FFST information to multiple FFST files. If multiple errors occur during a single execution of the application, each FFST record is written to a different FFST file.

Sections of an FFST record

An FFST record that is generated by the IBM MQ classes for JMS contains the following sections:

The header
A header, indicating the time when the FFST record was created, the platform that the IBM MQ classes for JMS application is running on, and the internal method that was being called. The header also contains a probe identifier, which uniquely identifies the place within the IBM MQ classes for JMS that generated the FFST record.

Data Some internal data that is associated with the FFST record.

Version information
Information about the version of the IBM MQ classes for JMS being used by the application that generated the FFST record.

Stack Trace
The Java stack trace for the thread that generated the FFST record.

Property Store Contents
A list of all of the Java system properties that have been set on the Java Runtime Environment that the IBM MQ classes for JMS application is running in.

WorkQueueMananger Contents
Information about the internal thread pool that is used by the IBM MQ classes for JMS .

Runtime properties
Details about the amount of memory and the number of processors available on the system where the IBM MQ classes for JMS application is running.

Component Manager Contents
Some information about the internal components that are loaded by the IBM MQ classes for JMS .

Provider Specific information
Information about all of the active JMS Connections, JMS Sessions, MessageProducer, and MessageConsumer objects currently being used by the IBM MQ classes for JMS application that was running when the FFST was generated. This information includes the name of the queue manager that JMS Connections and JMS Sessions are connected to, and the name of the IBM MQ queue or topic objects that are being used by MessageProducers and MessageConsumers.

All Thread information
Details about the state of all of the active threads in the Java Runtime Environment that the IBM MQ classes for JMS application was running in when the FFST record was generated. The name of each thread is shown, together with a Java stack trace for every thread.

Example FFST log file

A typical FFST log is shown in Figure 116 on page 1669

```
----------------------------------START FFST------------------------------------
c:\JBoss-6.0.0\bin\FFDC\JMSCC0007.FDC PID:4472

JMS Common Client First Failure Symptom Report


Product      :- IBM MQ classes for JMS
Date/Time     :- Mon Feb 03 14:14:46 GMT 2014
System time   :- 1391436886081
Operating System :- Windows Server 2008
UserID       :- pault
Java Vendor   :- IBM Corporation
Java Version   :- 2.6

Source Class   :- com.ibm.msg.client.commonservices.j2se.wmqsupport.PropertyStoreImpl
Source Method  :- getBooleanProperty(String)
ProbeID      :- XS002005
Thread       :- name=pool-1-thread-3 priority=5 group=workmanager-threads
ccl=BaseClassLoader@ef1c3794{vfs:///C:/JBoss-6.0.0/server/default/deploy/basicMDB.ear}

Data
----

|  name :- com.ibm.mq.connector.performJavaEEContainerChecks

Version information
-------------------

Java Message Service Client
7.5.0.2
p750-002-130627
Production

IBM MQ classes for Java Message Service
7.5.0.2
p750-002-130627
Production

IBM MQ JMS Provider
7.5.0.2
p750-002-130627
Production

Common Services for Java Platform, Standard Edition
7.5.0.2
p750-002-130627
Production



Stack trace
-----------

Stack trace to show the location of the FFST call
|  FFST Location :- java.lang.Exception
|   at com.ibm.msg.client.commonservices.trace.Trace.getCurrentPosition(Trace.java:1972)
|   at com.ibm.msg.client.commonservices.trace.Trace.createFFSTString(Trace.java:1911)
|   at com.ibm.msg.client.commonservices.trace.Trace.ffstInternal(Trace.java:1800)
|   at com.ibm.msg.client.commonservices.trace.Trace.ffst(Trace.java:1624)
|   at com.ibm.msg.client.commonservices.j2se.propertystore.PropertyStoreImpl.getBooleanProperty(
PropertyStoreImpl.java:322)
|   at com.ibm.msg.client.commonservices.propertystore.PropertyStore.getBooleanPropertyObject(Pr
opertyStore.java:302)
|   at com.ibm.mq.connector.outbound.ConnectionWrapper.jcaMethodAllowed(ConnectionWrapper.java:510)
|   at com.ibm.mq.connector.outbound.ConnectionWrapper.setExceptionListener(ConnectionWrapper.java:244)
|   at com.ibm.basicMDB.MDB.onMessage(MDB.java:45)
...

Property Store Contents
-----------------------

All currently set properties
|  awt.toolkit                           :- sun.awt.windows.WToolkit
|  catalina.ext.dirs                        :- C:\JBoss-6.0.0\server\default\lib
|  catalina.home                          :- C:\JBoss-6.0.0\server\default
|  com.ibm.cpu.endian                        :- little
|  com.ibm.jcl.checkClassPath                    :-
|  com.ibm.mq.connector.performJavaEEContainerChecks        :- false
|  com.ibm.oti.configuration                    :- scar
|  com.ibm.oti.jcl.build                      :- 20131013_170512
|  com.ibm.oti.shared.enabled                    :- false
|  com.ibm.oti.vm.bootstrap.library.path              :- C:\Program
Files\IBM\Java70\jre\bin\compressedrefs;C:\Program Files\IBM\Java70\jre\bin
|  com.ibm.oti.vm.library.version                  :- 26
|  com.ibm.system.agent.path                    :- C:\Program
Files\IBM\Java70\jre\bin
|  com.ibm.util.extralibs.properties                 :-
|  com.ibm.vm.bitmode                        :- 64
|  com.ibm.zero.version                       :- 2
|  console.encoding                        :- Cp850
|  file.encoding                          :- Cp1252
|  file.encoding.pkg                        :- sun.io
...


WorkQueueMananger Contents
--------------------------

|  Current ThreadPool size   :- 2
|  Maintain ThreadPool size   :- false
|  Maximum ThreadPool size   :- -1
|  ThreadPool inactive timeout :- 0

Runtime properties
------------------

|  Available processors     :- 4
|  Free memory in bytes (now)  :- 54674936
|  Max memory in bytes     :- 536870912
|  Total memory in bytes (now) :- 235012096

Component Manager Contents
--------------------------

Common Services Components:
|  CMVC        :- p750-002-130627
|  Class Name     :- class com.ibm.msg.client.commonservices.j2se.J2SEComponent
|  Component Name   :- com.ibm.msg.client.commonservices.j2se
|  Component Title  :- Common Services for Java Platform, Standard Edition
|  Factory Class   :- class com.ibm.msg.client.commonservices.j2se.CommonServicesImplementation
|  Version      :- 7.5.0.2
|  inPreferenceTo[0] :- com.ibm.msg.client.commonservices.j2me

Messaging Provider Components:
|  CMVC       :- p750-002-130627
|  Class Name    :- class com.ibm.msg.client.wmq.factories.WMQComponent
|  Component Name  :- com.ibm.msg.client.wmq
|  Component Title :- IBM MQ JMS Provider
|  Factory Class  :- class com.ibm.msg.client.wmq.factories.WMQFactoryFactory
|  Version     :- 7.5.0.2
```

```
Provider Specific Information
----------------------------

Overview of JMS System
Num. Connections : 3
```

The information in the header, Data, and Stack Trace sections of the FFST record are used by IBM to assist in problem determination. In many cases, there is little that the system administrator can do when an FFST record is generated, apart from raising problems through the IBM Support Center.

Suppressing FFST records

An FFST file that is generated by the IBM MQ classes for JMS contain one FFST record. If a problem occurs multiple times during the execution of an IBM MQ classes for JMS application, multiple FFST files with the same probe identifier are generated. This might not be desirable. The property com.ibm.msg.client.commonservices.ffst.suppress can be used to suppress the production of FFST files. This property must be set in the IBM MQ classes for JMS configuration file used by the application, and can take the following values:

0: Output all FFDC files (default).

-1: Output only the first FFST file for a probe identifier.

integer: Suppress all FFST files for a probe identifier except those files that are a multiple of this number.

FFST: IBM MQ for Windows

Describes the name, location, and contents of the First Failure Support Technology (FFST) files for Windows systems.

In IBM MQ for Windows, FFST information is recorded in a file in the `C:\Program Files\IBM\WebSphere MQ\errors` directory.

An FFST file contains one or more records. Each FFST record contains information about an error that is normally severe, and possibly unrecoverable. These records typically indicate either a configuration problem with the system or an IBM MQ internal error.

FFST files are named `AMQ` *`nnnnn.mm`*`.FDC`, where:

nnnnn
: Is the ID of the process reporting the error

mm
: Starts at 0. If the full file name already exists, this value is incremented by one until a unique FFST file name is found. An FFST file name can already exist if a process is reused.

An instance of a process will write all FFST information to the same FFST file. If multiple errors occur during a single execution of the process, an FFST file can contain many records.

When a process writes an FFST record it also sends a record to the Event Log. The record contains the name of the FFST file to assist in automatic problem tracking. The Event log entry is made at the application level.

A typical FFST log is shown in Figure 117 on page 1671.

```
+-----------------------------------------------------------------------------+
| WebSphere MQ First Failure Symptom Report                                   |
| =========================================                                   |
|                                                                             |
| Date/Time         :- Mon January 28 2008 21:59:06 GMT                      |
| UTC Time/Zone     :- 1201539869.892015 0 GMT                               |
| Host Name         :- 99VXY09 (Windows XP Build 2600: Service Pack 1)       |
| PIDS              :- 5724H7200                                             |
| LVLS              :- 7.0.0.0                                               |
| Product Long Name :- WebSphere MQ for Windows                              |
| Vendor            :- IBM                                                   |
| Probe Id          :- HL010004                                              |
| Application Name  :- MQM                                                   |
| Component         :- hlgReserveLogSpace                                    |
| SCCS Info         :- lib/logger/amqhlge0.c, 1.26                           |
| Line Number       :- 246                                                   |
| Build Date        :- Jan 25 2008                                           |
| CMVC level        :- p000-L050202                                          |
| Build Type        :- IKAP - (Production)                                   |
| UserID            :- IBM_User                                              |
| Process Name      :- C:\Program Files\IBM\WebSphere MQ\bin\amqzlaa0.exe    |
| Process           :- 00003456                                              |
| Thread            :- 00000030                                              |
| QueueManager      :- qmgr2                                                 |
| ConnId(1) IPCC    :- 162                                                   |
| ConnId(2) QM      :- 45                                                    |
| Major Errorcode   :- hrcE_LOG_FULL                                         |
| Minor Errorcode   :- OK                                                    |
| Probe Type        :- MSGAMQ6709                                            |
| Probe Severity    :- 2                                                     |
| Probe Description :- AMQ6709: The log for the Queue manager is full.       |
| FDCSequenceNumber :- 0                                                     |
+-----------------------------------------------------------------------------+

MQM Function Stack
zlaMainThread
zlaProcessMessage
zlaProcessMQIRequest
zlaMQPUT
zsqMQPUT
kpiMQPUT
kqiPutIt
kqiPutMsgSegments
apiPutMessage
aqmPutMessage
aqhPutMessage
aqqWriteMsg
aqqWriteMsgData
aqlReservePutSpace
almReserveSpace
hlgReserveLogSpace
xcsFFST

MQM Trace History
-------------} hlgReserveLogSpace rc=hrcW_LOG_GETTING_VERY_FULL
-------------{ xllLongLockRequest
-------------} xllLongLockRequest rc=OK

...
```

Figure 117. Sample IBM MQ for Windows First Failure Symptom Report

The Function Stack and Trace History are used by IBM to assist in problem determination. In many cases there is little that the system administrator can do when an FFST record is generated, apart from raising problems through the IBM Support Center.

In certain circumstances a small dump file can be generated in addition to an FFST file and placed in the `C:\Program Files\IBM\WebSphere MQ\errors` directory. A dump file will have the same name as the FFST file, in the form `AMQnnnnn.mm.dmp`. These files can be used by IBM to assist in problem determination.

First Failure Support Technology (FFST) files and Windows clients

The files are produced already formatted and are in the errors subdirectory of the IBM MQ MQI client installation directory.

These are normally severe, unrecoverable errors and indicate either a configuration problem with the system or an IBM MQ internal error.

The files are named `AMQnnnnn.mm.FDC`, where:
- `nnnnn` is the process ID reporting the error
- `mm` is a sequence number, normally 0

When a process creates an FFST it also sends a record to the system log. The record contains the name of the FFST file to assist in automatic problem tracking.

The system log entry is made at the “user.error” level.

First Failure Support Technology is explained in detail in First Failure Support Technology (FFST).

FFST: IBM MQ for UNIX and Linux systems

Describes the name, location, and contents of the First Failure Support Technology (FFST) files for UNIX and Linux systems.

For IBM MQ on UNIX and Linux systems, FFST information is recorded in a file in the `/var/mqm/errors` directory.

An FFST file contains one or more records. Each FFST record contains information about an error that is normally severe, and possibly unrecoverable. These records indicate either a configuration problem with the system or an IBM MQ internal error.

FFST files are named `AMQ` *nnnnn.mm*`.FDC`, where:

nnnnn
: Is the ID of the process reporting the error

mm
: Starts at 0. If the full file name already exists, this value is incremented by one until a unique FFST file name is found. An FFST file name can already exist if a process is reused.

An instance of a process will write all FFST information to the same FFST file. If multiple errors occur during a single execution of the process, an FFST file can contain many records.

In order to read the contents of a FFST file, you must be either the creator of the file, or a member of the mqm group.

When a process writes an FFST record, it also sends a record to syslog. The record contains the name of the FFST file to assist in automatic problem tracking. The syslog entry is made at the *user.error* level. See the operating-system documentation about `syslog.conf` for information about configuring this.

Some typical FFST data is shown in Figure 118 on page 1673.

```
+-----------------------------------------------------------------------------+
|                                                                             |
| WebSphere MQ First Failure Symptom Report                                   |
| =========================================                                   |
|                                                                             |
| Date/Time         :- Mon January 28 2008 21:59:06 GMT                       |
| UTC Time/Zone     :- 1201539869.892015 0 GMT                                |
| Host Name         :- mqperfh2 (HP-UX B.11.23)                               |
| PIDS              :- 5724H7202                                              |
| LVLS              :- 7.0.0.0                                                |
| Product Long Name :- WebSphere MQ for HP-UX                                 |
| Vendor            :- IBM                                                    |
| Probe Id          :- XC034255                                               |
| Application Name  :- MQM                                                    |
| Component         :- xcsWaitEventSem                                        |
| SCCS Info         :- lib/cs/unix/amqxerrx.c, 1.204                          |
| Line Number       :- 6262                                                   |
| Build Date        :- Jan 25 2008                                            |
| CMVC level        :- p000-L050203                                           |
| Build Type        :- IKAP - (Production)                                    |
| UserID            :- 00000106 (mqperf)                                      |
| Program Name      :- amqzmuc0                                               |
| Addressing mode   :- 64-bit                                                 |
| Process           :- 15497                                                  |
| Thread            :- 1                                                      |
| QueueManager      :- CSIM                                                   |
| ConnId(2) QM      :- 4                                                      |
| Major Errorcode   :- OK                                                     |
| Minor Errorcode   :- OK                                                     |
| Probe Type        :- INCORROUT                                              |
| Probe Severity    :- 4                                                      |
| Probe Description :- AMQ6109: An internal WebSphere MQ error has occurred.  |
| FDCSequenceNumber :- 0                                                      |
|                                                                             |
+-----------------------------------------------------------------------------+

MQM Function Stack
amqzmuc0
xcsWaitEventSem
xcsFFST

MQM Trace History
Data: 0x00003c87
--} xcsCheckProcess rc=OK
--{ xcsRequestMutexSem
--} xcsRequestMutexSem rc=OK

...
```

Figure 118. FFST report for IBM MQ for UNIX systems

The Function Stack and Trace History are used by IBM to assist in problem determination. In many cases there is little that the system administrator can do when an FFST report is generated, apart from raising problems through the IBM Support Center.

However, there are some problems that the system administrator might be able to solve. If the FFST shows *out of resource* or *out of space on device* descriptions when calling one of the IPC functions (for example, **semop** or **shmget**), it is likely that the relevant kernel parameter limit has been exceeded.

If the FFST report shows a problem with **setitimer** , it is likely that a change to the kernel timer parameters is needed.

To resolve these problems, increase the IPC limits, rebuild the kernel, and restart the machine.

First Failure Support Technology (FFST) files and UNIX and Linux clients

FFST logs are written when a severe IBM MQ error occurs. They are written to the directory `/var/mqm/errors`.

These are normally severe, unrecoverable errors and indicate either a configuration problem with the system or an IBM MQ internal error.

The files are named `AMQnnnnn.mm.FDC`, where:
- `nnnnn` is the process id reporting the error
- `mm` is a sequence number, normally 0

When a process creates an FFST it also sends a record to the system log. The record contains the name of the FFST file to assist in automatic problem tracking.

The system log entry is made at the "user.error" level.

First Failure Support Technology is explained in detail in First Failure Support Technology (FFST).

FFST: IBM MQ for IBM i

Describes the name, location, and contents of the First Failure Support Technology (FFST) files for IBM i systems.

For IBM i, FFST information is recorded in a stream file in the `/QIBM/UserData/mqm/errors` directory.

These errors are normally severe, unrecoverable errors, and indicate either a configuration problem with the system or an IBM MQ internal error.

The stream files are named `AMQ` *`nnnnn.mm`*`.FDC`, where:

nnnnn	Is the ID of the process reporting the error
mm	Is a sequence number, normally 0

A copy of the job log of the failing job is written to a file with the same name as the .FDC file. The file name ends with .JOB.

Some typical FFST data is shown in the following example.

```
-------------------------------------------------------------------------------
| WebSphere MQ First Failure Symptom Report                                   |
| =========================================                                   |
|                                                                             |
| Date/Time         :- Mon January 28 2008 21:59:06 GMT                       |
| UTC Time/Zone     :- 1201539869.892015 0 GMT                                |
| Host Name         :- WINAS12B.HURSLEY.IBM.COM                               |
| PIDS              :- 5733A38                                                |
| LVLS              :- 520                                                    |
| Product Long Name :- WebSphere MQ for IBMi                                  |
| Vendor            :- IBM                                                    |
| Probe Id          :- XY353001                                               |
| Application Name  :- MQM                                                    |
| Component         :- xehAS400ConditionHandler                               |
| Build Date        :- Feb 25 2008                                            |
| UserID            :- 00000331 (MAYFCT)                                      |
| Program Name      :- STRMQM_R  MAYFCT                                       |
| Job Name          :- 020100/MAYFCT/STRMQM_R                                 |
| Activation Group  :- 101 (QMQM) (QMQM/STRMQM_R)                             |
| Process           :- 00001689                                               |
| Thread            :- 00000001                                               |
| QueueManager      :- TEST.AS400.OE.P                                         |
| Major Errorcode   :- STOP                                                    |
| Minor Errorcode   :- OK                                                      |
| Probe Type        :- HALT6109                                                |
| Probe Severity    :- 1                                                       |
| Probe Description :- 0                                                       |
| Arith1            :- 1 1                                                     |
| Comment1          :- 00d0                                                    |
------------------------------------------------------------------------------

MQM Function Stack
lpiSPIMQConnect
zstMQConnect
ziiMQCONN
ziiClearUpAgent
xcsTerminate
xlsThreadInitialization
xcsConnectSharedMem
xstConnSetInSPbyHandle
xstConnSharedMemSet
xcsFFST

MQM Trace History
<-- xcsCheckProcess rc=xecP_E_INVALID_PID
-->
xcsCheckProcess
<-- xcsCheckProcess rc=xecP_E_INVALID_PID
-->
xlsThreadInitialization
-->
xcsConnectSharedMem
-->
xcsRequestThreadMutexSem
<-- xcsRequestThreadMutexSem rc=OK
-->
xihGetConnSPDetailsFromList
<-- xihGetConnSPDetailsFromList rc=OK
-->
xstCreateConnExtentList
<-- xstCreateConnExtentList rc=OK
-->
xstConnSetInSPbyHandle
-->
xstSerialiseSPList
-->
xllSpinLockRequest
<-- xllSpinLockRequest rc=OK
<-- xstSerialiseSPList rc=OK
-->
xstGetSetDetailsFromSPByHandle
<-- xstGetSetDetailsFromSPByHandle rc=OK
-->
xstConnSharedMemSet
-->
xstConnectExtent
-->
xstAddConnExtentToList
<-- xstAddConnExtentToList rc=OK
<-- xstConnectExtent rc=OK
-->
xcsBuildDumpPtr
-->
xcsGetMem
<-- xcsGetMem rc=OK
<-- xcsBuildDumpPtr rc=OK
-->
xcsBuildDumpPtr
<-- xcsBuildDumpPtr rc=OK
-->
xcsBuildDumpPtr
<-- xcsBuildDumpPtr rc=OK
-->
xcsFFST

Process Control Block
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :8bba0:0:6d  E7C9C8D7 000004E0 00000699 00000000  XIHP...\...r....
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :8bbb0:1:6d  00000000 00000002 00000000 00000000  ................
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :8bbc0:2:6d  80000000 00000000 EC161F7C FC002DB0  ...........@...¢
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :8bbd0:3:6d  80000000 00000000 EC161F7C FC002DB0  ...........@...¢
```

```
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :8bbe0:4:6d  00000000 00000000 00000000 00000000  ................

Thread Control Block
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :1db0:20:6d  E7C9C8E3 00001320 00000000 00000000  XIHT............
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :1dc0:21:6d  00000001 00000000 00000000 00000000  ................
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :1dd0:22:6d  80000000 00000000 DD13C17B 81001000  ..........A#a...
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :1de0:23:6d  00000000 00000046 00000002 00000001  ................
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :1df0:24:6d  00000000 00000000 00000000 00000000  ................


RecoveryIndex
SPP:0000 :1aefSTRMQM_R MAYFCT  020100 :2064:128:6d  00000000                 ....
```

Note:

1. The `MQM Trace History` section is a log of the 200 most recent function trace statements, and is recorded in the FFST report regardless of any TRCMQM settings.
2. The queue manager details are recorded only for jobs that are connected to a queue manager subpool.
3. When the failing component is `xehAS400ConditionHandler`, additional data is logged in the errors directory giving extracts from the job log relating to the exception condition.

The function stack and trace history are used by IBM to assist in problem determination. In most cases, there is little that the system administrator can do when an FFST report is generated, apart from raising problems through the IBM Support Center.

FFST: IBM MQ for HP Integrity NonStop Server

Describes the name, location, and contents of the First Failure Support Technology™ (FFST™) files for HP Integrity NonStop Server systems.

In IBM MQ client for HP Integrity NonStop Server systems, FFST information is recorded in a file in the `<mqpath>/var/mqm/errors` directory.

An FFST file contains one or more records. Each FFST record contains information about an error that is normally severe, and possibly unrecoverable. These records indicate either a configuration problem with the system or an IBM MQ internal error.

FFST files are named `AMQ.nnn.xx.ppp.qq.FDC`, where:

nnn The name of the process that is reporting the error.

xx The processor number on which the process is running.

ppp The PIN of the process that you are tracing.

qq A sequence that starts at 0. If the full file name exists, this value is incremented by one until a unique FFST file name is found. An FFST file name can exist if a process is reused.

Each field can contain fewer or more digits than shown in the example.

An instance of a process writes all FFST information to the same FFST file. If multiple errors occur during a single execution of the process, an FFST file can contain many records.

To read the contents of an FFST file, you must be either the creator of the file, or a member of the mqm group.

When a process writes an FFST record, it also creates an EMS event.

Figure 119 on page 1677 shows a typical FFST report for an IBM MQ client on a HP Integrity NonStop Server system:

```
+-----------------------------------------------------------------------------+
|                                                                             |
| WebSphere MQ First Failure Symptom Report                                   |
| =========================================                                   |
|                                                                             |
| Date/Time         :- Mon April 29 2013 10:21:26 EDT                         |
| UTC Time          :- 1367245286.105303                                      |
| UTC Time Offset   :- -240 (EST)                                             |
| Host Name         :- MYHOST                                                 |
| Operating System  :- HP NonStop J06.14, NSE-AB 069194                       |
|                                                                             |
| PIDS              :- 5724H7222                                              |
| LVLS              :- 7.1.0.0                                                |
| Product Long Name :- WebSphere MQ for HP NonStop Server                     |
| Vendor            :- IBM                                                    |
| Installation Path :- /home/cmarti/client/opt/mqm                            |
| Probe Id          :- MQ000020                                               |
| Application Name  :- MQM                                                    |
| Component         :- Unknown                                                |
| SCCS Info         :- S:/cmd/trace/amqxdspa.c,                               |
| Line Number       :- 3374                                                   |
| Build Date        :- Apr 24 2013                                            |
| Build Level       :- D20130424-1027                                         |
| Build Type        :- ICOL - (Development)                                   |
| File Descriptor   :- 6                                                      |
| Effective UserID  :- 11329 (MQM.CMARTI)                                     |
| Real UserID       :- 11329 (MQM.CMARTI)                                     |
| Program Name      :- dspmqtrc                                               |
| Addressing mode   :- 32-bit                                                 |
| LANG              :-                                                        |
| Process           :- 1,656 $Y376 OSS 469762429                              |
| Thread(n)         :- 1                                                      |
| UserApp           :- FALSE                                                  |
| Last HQC          :- 0.0.0-0                                                |
| Last HSHMEMB      :- 0.0.0-0                                                |
| Major Errorcode   :- krcE_UNEXPECTED_ERROR                                  |
| Minor Errorcode   :- OK                                                     |
| Probe Type        :- INCORROUT                                              |
| Probe Severity    :- 2                                                      |
| Probe Description :- AMQ6125: An internal WebSphere MQ error has occurred.  |
| FDCSequenceNumber :- 0                                                      |
| Comment1          :- AMQ.3.520.sq_tc.0.TRC                                  |
| Comment2          :- Unrecognised hookID:0x3 at file offset 0x4b84          |
|                                                                             |
+-----------------------------------------------------------------------------+

MQM Function Stack
xcsFFST

MQM Trace History
{ xppInitialiseDestructorRegistrations
} xppInitialiseDestructorRegistrations rc=OK
{ xcsGetEnvironmentInteger
-{ xcsGetEnvironmentString

...
```

Figure 119. Sample FFST data

The Function Stack and Trace History are used by IBM to help problem determination. In many cases, there is little that the system administrator can do when an FFST report is generated, apart from raising problems through the IBM Support Center. However, there are some problems that the system administrator might be able to solve, for example, if the FFST report shows `Out of resource` or `Out of space on device`.

For more information about FFST, see “First Failure Support Technology (FFST” on page 1666.

Using logs

There are a variety of logs that you can use to help with problem determination and troubleshooting.

Use the following links to find out about the logs available for your platform and how to use them:

- Windows UNIX Linux “Error logs on Windows, UNIX and Linux systems”
- IBM i “Error logs on IBM i” on page 1683
- “Error logs on HP Integrity NonStop Server” on page 1682

z/OS On z/OS error messages are written to:

- The z/OS system console
- The channel-initiator job log

It is possible to suppress or exclude some messages on both distributed and z/OS systems.

For details of suppressing some messages on distributed systems, see Queue manager error logs.

z/OS On z/OS, if you are using the z/OS message processing facility to suppress messages, the console messages can be suppressed. See IBM MQ for z/OS concepts for more information.

z/OS For information about error messages, console logs, and dumps on IBM MQ for z/OS, see Problem determination on z/OS.

Related concepts:

“Troubleshooting and support” on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

“Troubleshooting overview” on page 1276
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

“First Failure Support Technology (FFST” on page 1666
First Failure Support Technology (FFST) for IBM MQ provides information about events that, in the case of an error, can help IBM support personnel to diagnose the problem.

Related tasks:

“Using trace” on page 1686
You can use different types of trace to help you with problem determination and troubleshooting.

Error logs on Windows, UNIX and Linux systems

About error log files, and an example.

At installation time, an `errors` subdirectory is created in the `/var/mqm` file path under UNIX and Linux systems, and in the installation directory, for example `C:\Program Files\IBM\WebSphere MQ\` file path under Windows systems. The `errors` subdirectory can contain up to three error log files named:

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

For more information about directories where log files are stored, see “Error log directories” on page 1681.

After you have created a queue manager, it creates three error log files when it needs them. These files have the same names as those files in the system error log directory. That is, AMQERR01, AMQERR02, and AMQERR03, and each has a default capacity of 2 MB (2 097 152 bytes). The capacity can be altered in the `Extended` queue manager properties page from the IBM MQ Explorer, or in the `QMErrorLog` stanza in the qm.ini file. These files are placed in the `errors` subdirectory in the queue manager data directory that you selected when you installed IBM MQ or created your queue manager. The default location for the `errors` subdirectory is `/var/mqm/qmgrs/` *qmname* file path under UNIX and Linux systems, and `C:\Program Files\IBM\WebSphere MQ\qmgrs\` *qmname* `\errors` file path under Windows systems.

As error messages are generated, they are placed in AMQERR01. When AMQERR01 gets bigger than 2 MB (2 097 152 bytes) it is copied to AMQERR02. Before the copy, AMQERR02 is copied to AMQERR03.LOG. The previous contents, if any, of AMQERR03 are discarded.

The latest error messages are thus always placed in AMQERR01, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate error files belonging to the queue manager, unless the queue manager is unavailable, or its name is unknown. In which case, channel-related messages are placed in the system error log directory.

To examine the contents of any error log file, use your usual system editor.

An example of an error log

Figure 120 shows an extract from an IBM MQ error log:

```
17/11/2004 10:32:29 - Process(2132.1) User(USER_1) Program(runmqchi.exe)
Host(HOST_1) Installation(Installation1)
VRMF(7.1.0.0) QMgr (A.B.C)
AMQ9542: Queue manager is ending.

EXPLANATION:
The program will end because the queue manager is quiescing.
ACTION:
None.
----- amqrimna.c : 931 -------------------------------------------------------
```

Figure 120. Sample IBM MQ error log

Operator messages

Operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. Operator messages are national-language enabled, with message catalogs installed in standard locations.

These messages are written to the associated window, if any. In addition, some operator messages are written to the AMQERR01.LOG file in the queue manager directory, and others to the equivalent file in the system error log directory.

Error log access restrictions

Certain error log directories and error logs have access restrictions.

To gain the following access permissions, a user or application must be a member of the mqm group:

- Read and write access to all queue manager error log directories.
- Read and write access to all queue manager error logs.
- Write access to the system error logs.

If an unauthorized user or application attempts to write a message to a queue manager error log directory, the message is redirected to the system error log directory.

Ignoring error codes under UNIX and Linux systems

On UNIX and Linux systems, if you do not want certain error messages to be written to a queue manager error log, you can specify the error codes that are to be ignored using the QMErrorLog stanza.

For more information, see Queue manager error logs.

Ignoring error codes under Windows systems

On Windows systems, the error message is written to both the IBM MQ error log and the Windows Application Event Log. The error messages written to the Application Event Log includes messages of error severity, warning severity, and information severity. If you do not want certain error messages to be written to the Windows Application Event Log, you can specify the error codes that are to be ignored in the Windows registry.

Use the following registry key:

```
HKLM\Software\IBM\WebSphere MQ\Installation\MQ_INSTALLATION_NAME\IgnoredErrorCodes
```

where *MQ_INSTALLATION_NAME* is the installation name associated with a particular installation of IBM MQ.

The value that you set it to is an array of strings delimited by the NULL character, with each string value relating to the error code that you want ignored from the error log. The complete list is terminated with a NULL character, which is of type REG_MULTI_SZ.

For example, if you want IBM MQ to exclude error codes AMQ3045, AMQ6055, and AMQ8079 from the Windows Application Event Log, set the value to:

```
AMQ3045\0AMQ6055\0AMQ8079\0\0
```

The list of messages you want to exclude is defined for all queue managers on the machine. Any changes you make to the configuration will not take effect until each queue manager is restarted.

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Using logs" on page 1678
There are a variety of logs that you can use to help with problem determination and troubleshooting.

z/OS "Problem determination on z/OS" on page 1724
IBM MQ for z/OS, CICS, Db2, and IMS produce diagnostic information which can be used for problem determination.

Related tasks:

"Using trace" on page 1686
You can use different types of trace to help you with problem determination and troubleshooting.

Related reference:

IBM i "Error logs on IBM i" on page 1683
Use this information to understand the IBM MQ for IBM i error logs.

Error log directories

IBM MQ uses a number of error logs to capture messages concerning its own operation of IBM MQ, any queue managers that you start, and error data coming from the channels that are in use. The location of the error logs depends on whether the queue manager name is known and whether the error is associated with a client.

The location the error logs are stored in depends on whether the queue manager name is known and whether the error is associated with a client. *MQ_INSTALLATION_PATH* represents the high level directory where IBM MQ is installed.

- If the queue manager name is known, the location of the error log is shown in Table 156.

Table 156. Queue manager error log directory

Platform	Directory
UNIX and Linux systems	`/var/mqm/qmgrs/` *qmname* `/errors`
Windows systems	*MQ_INSTALLATION_PATH*`\QMGRS\` *qmname* `\ERRORS\AMQERR01.LOG`

- If the queue manager name is not known, the location of the error log is shown in Table 157.

Table 157. System error log directory

Platform	Directory
UNIX and Linux systems	`/var/mqm/errors`
Windows systems	*MQ_INSTALLATION_PATH*`\QMGRS\@SYSTEM\ERRORS\AMQERR01.LOG`

- If an error has occurred with a client application, the location of the error log on the client is shown in Table 158.

Table 158. Client error log directory

Platform	Directory
UNIX and Linux systems	`/var/mqm/errors`
Windows systems	`MQ_DATA_PATH\ERRORS\AMQERR01.LOG`

In IBM MQ for Windows, an indication of the error is also added to the Application Log, which can be examined with the Event Viewer application provided with Windows systems.

Early errors

There are a number of special cases where these error logs have not yet been established and an error occurs. IBM MQ attempts to record any such errors in an error log. The location of the log depends on how much of a queue manager has been established.

If, because of a corrupt configuration file for example, no location information can be determined, errors are logged to an errors directory that is created at installation time on the root directory (`/var/mqm` or `C:\Program Files\IBM\WebSphere MQ`).

If IBM MQ can read its configuration information, and can access the value for the Default Prefix, errors are logged in the errors subdirectory of the directory identified by the Default Prefix attribute. For example, if the default prefix is `C:\Program Files\IBM\WebSphere MQ`, errors are logged in `C:\Program Files\IBM\WebSphere MQ\errors`.

For further information about configuration files, see Changing IBM MQ and queue manager configuration information.

Note: Errors in the Windows Registry are notified by messages when a queue manager is started.

Error logs on HP Integrity NonStop Server

Use this information to understand the IBM MQ client on HP Integrity NonStop Server error logs, together with an example.

At installation time, an errors subdirectory is created in the `<mqpath>/var/mqm` file path. The errors subdirectory can contain up to three error log files named:

- `AMQERR01.LOG`
- `AMQERR02.LOG`
- `AMQERR03.LOG`

As error messages are generated, they are written to `AMQERR01.LOG`. When `AMQERR01.LOG` gets bigger than 2 MB (2 097 152 bytes), it is copied to `AMQERR02.LOG`. Before the copy, `AMQERR02.LOG` is copied to `AMQERR03.LOG`. The previous contents, if any, of `AMQERR03.LOG` are discarded.

The latest error messages are therefore always placed in `AMQERR01.LOG`. The other log files are used to maintain a history of error messages.

To examine the contents of any error log file, use your system editor. The contents of the log files can read by any user, but write access requires the user to be a member of the mqm group.

An example of an error log

Figure 121 shows an extract from an IBM MQ error log:

```
04/30/13 06:18:22 - Process(320406477.1) User(MYUSER) Program(nssfcps_c)
                    Host(myhost)
                    VRMF(7.1.0.0)
AMQ9558: The remote channel 'SYSTEM.DEF.SVRCONN' on host 'hostname
(x.x.x.x)(1414)' is not currently available.

EXPLANATION:
The channel program ended because an instance of channel 'SYSTEM.DEF.SVRCONN'
could not be started on the remote system. This could be for one of the
following reasons:

The channel is disabled.

The remote system does not have sufficient resources to run another instance of
the channel.

In the case of a client-connection channel, the limit on the number of
instances configured for the remote server-connection channel was reached.

ACTION:
Check the remote system to ensure that the channel is able to run. Try the
operation again.
----- cmqxrfpt.c : 504 --------------------------------------------------------
```

Figure 121. Sample IBM MQ error log

Error logs on IBM i

IBM i

Use this information to understand the IBM MQ for IBM i error logs.

By default, only members of the QMQMADM group can access error logs. To give users access to error logs, who are not members of this group, set **`ValidateAuth`** to *No* and grant those users *PUBLIC authority. See Filesystem for more information.

IBM MQ uses a number of error logs to capture messages concerning the operation of IBM MQ itself, any queue managers that you start, and error data coming from the channels that are in use.

At installation time, a `/QIBM/UserData/mqm/errors` subdirectory is created in the IFS.

The location of the error logs depends on whether the queue manager name is known.

In the IFS:
- If the queue manager name is known and the queue manager is available, error logs are located in:
  `/QIBM/UserData/mqm/qmgrs/`*`qmname`*`/errors`
- If the queue manager is not available, error logs are located in:
  `/QIBM/UserData/mqm/errors`

You can use the system utility EDTF to browse the errors directories and files. For example:

```
EDTF '/QIBM/UserData/mqm/errors'
```

Alternatively, you can use option 23 against the queue manager from the WRKMQM panel.

The errors subdirectory can contain up to three error log files named:
- `AMQERR01.LOG`
- `AMQERR02.LOG`
- `AMQERR03.LOG`

After you have created a queue manager, three error log files are created when they are needed by the queue manager. These files have the same names as the `/QIBM/UserData/mqm/errors` ones, that is AMQERR01, AMQERR02, and AMQERR03, and each has a capacity of 2 MB (2 097 152 bytes). The files are placed in the errors subdirectory of each queue manager that you create, that is `/QIBM/UserData/mqm/qmgrs/`*`qmname`*`/errors`.

As error messages are generated, they are placed in AMQERR01. When AMQERR01 gets bigger than 2 MB (2 097 152 bytes), it is copied to AMQERR02. Before the copy, AMQERR02 is copied to AMQERR03.LOG. The previous contents, if any, of AMQERR03 are discarded.

The latest error messages are thus always placed in AMQERR01, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate errors files of the queue manager, unless the name of their queue manager is unknown or the queue manager is unavailable. When the queue manager name is unavailable or its name cannot be determined, channel-related messages are placed in the `/QIBM/UserData/mqm/errors` subdirectory.

To examine the contents of any error log file, use your system editor, EDTF, to view the stream files in the IFS.

Note:
1. Do not change ownership of these error logs.
2. If any error log file is deleted, it is automatically re-created when the next error message is logged.

Early errors

There are a number of special cases where the error logs have not yet been established and an error occurs. IBM MQ attempts to record any such errors in an error log. The location of the log depends on how much of a queue manager has been established.

If, because of a corrupted configuration file, for example, no location information can be determined, errors are logged to an errors directory that is created at installation time.

If both the IBM MQ configuration file and the DefaultPrefix attribute of the AllQueueManagers stanza are readable, errors are logged in the errors subdirectory of the directory identified by the DefaultPrefix attribute.

Operator messages

Operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. Operator messages are national language enabled, with message catalogs installed in standard locations.

These messages are written to the job log, if any. In addition, some operator messages are written to the `AMQERR01.LOG` file in the queue manager directory, and others to the `/QIBM/UserData/mqm/errors` directory copy of the error log.

An example IBM MQ error log

Figure 122 on page 1685 shows a typical extract from an IBM MQ error log.

```
************Beginning of data**************
07/19/02  11:15:56 AMQ9411: Repository manager ended normally.

EXPLANATION:
Cause . . . . . :   The repository manager ended normally.
Recovery  . . . :   None.
Technical Description . . . . . . . . :   None.
-------------------------------------------------------------------------------
07/19/02  11:15:57 AMQ9542: Queue manager is ending.

EXPLANATION:
Cause . . . . . :   The program will end because the queue manager is quiescing.
Recovery  . . . :   None.
Technical Description . . . . . . . . :   None.
----- amqrimna.c : 773 --------------------------------------------------------

07/19/02  11:16:00 AMQ8004: WebSphere MQ queue manager 'mick' ended.
EXPLANATION:
Cause . . . . . :   WebSphere MQ queue manager 'mick' ended.
Recovery  . . . :   None.
Technical Description . . . . . . . . :   None.
-------------------------------------------------------------------------------
07/19/02  11:16:48 AMQ7163: WebSphere MQ job number 18429 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18429 the CCSID is 37. The job name is
                    582775/MQUSER/AMQZXMA0.
Recovery  . . . :   None
-------------------------------------------------------------------------------
07/19/02  11:16:49 AMQ7163: WebSphere MQ job number 18430 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18430 the CCSID is 0. The job name is
                    582776/MQUSER/AMQZFUMA.
Recovery  . . . :   None
-------------------------------------------------------------------------------
07/19/02  11:16:49 AMQ7163: WebSphere MQ job number 18431 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18431 the CCSID is 37. The job name is
                    582777/MQUSER/AMQZXMAX.
Recovery  . . . :   None
-------------------------------------------------------------------------------
07/19/02  11:16:50 AMQ7163: WebSphere MQ job number 18432 started.

EXPLANATION:
Cause . . . . . :   This job has started to perform work for Queue Manager
                    mick, The job's PID is 18432 the CCSID is 37. The job name is
                    582778/MQUSER/AMQALMPX.
Recovery  . . . :   None
-------------------------------------------------------------------------------
```

Figure 122. Extract from an IBM MQ error log

Related concepts:

"Error logs on Windows, UNIX and Linux systems" on page 1678
About error log files, and an example.

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Using logs" on page 1678
There are a variety of logs that you can use to help with problem determination and troubleshooting.

z/OS "Problem determination on z/OS" on page 1724
IBM MQ for z/OS, CICS, Db2, and IMS produce diagnostic information which can be used for problem determination.

Related tasks:

"Using trace"
You can use different types of trace to help you with problem determination and troubleshooting.

Using trace

You can use different types of trace to help you with problem determination and troubleshooting.

About this task

Use this information to find out about the different types of trace, and how to run trace for your platform.

- "Using trace on Windows"
- "Using trace on UNIX and Linux systems" on page 1688
- IBM i "Using trace on IBM MQ server on IBM i" on page 1692
- IBM i "Using trace on IBM MQ client on IBM i" on page 1695
- z/OS "Using trace for problem determination on z/OS" on page 1697
- "Tracing TLS and SSL: **runmqakm** and iKeyman and iKeycmd functions" on page 1708
- "Tracing IBM MQ classes for JMS applications" on page 1709
- "Tracing IBM MQ classes for Java applications" on page 1713
- "Tracing the IBM MQ Resource Adapter" on page 1717
- "Tracing additional IBM MQ Java components" on page 1718
- "Controlling trace in a running process by using IBM MQ classes for Java and IBM MQ classes for JMS" on page 1721

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 1276
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

"Using logs" on page 1678
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"First Failure Support Technology (FFST" on page 1666
First Failure Support Technology (FFST) for IBM MQ provides information about events that, in the case of an error, can help IBM support personnel to diagnose the problem.

Related tasks:

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Using trace on Windows

Use the **strmqtrc** and **endmqtrc** commands or the MQ Explorer interface to start and end tracing.

Windows uses the following commands for the client trace facility:

strmqtrc
  to start tracing

endmqtrc
  to end tracing

The output files are created in the `MQ_DATA_PATH/trace` directory.

Trace files on IBM MQ for Windows

Trace files are named `AMQ` *`ppppp. qq`*`.TRC` where the variables are:

ppppp
  The ID of the process reporting the error.

qq
  A sequence number, starting at 0. If the full file name exists, this value is incremented by one until a unique trace file name is found. A trace file name can exist if a process is reused.

Note:
1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

To format or view a trace file, you must be either the creator of the trace file, or a member of the mqm group.

SSL trace files have the names `AMQ.SSL.TRC` and `AMQ.SSL.TRC.1`. You cannot format SSL trace files; send them unchanged to IBM support.

How to start and stop a trace

Enable or modify tracing using the **`strmqtrc`** control command (see strmqtrc). To stop tracing, use the **`endmqtrc`** control command (see endmqtrc).

In IBM MQ for Windows systems, you can also start and stop tracing using the MQ Explorer, as follows:
1. Start the MQ Explorer from the **Start** menu.
2. In the Navigator View, right-click the **IBM MQ** tree node, and select **Trace...**. The Trace Dialog is displayed.
3. Click **Start** or **Stop** as appropriate.

Selective component tracing

Use the `-t` and `-x` options to control the amount of trace detail to record. By default, all trace points are enabled. You can specify the points that you do not want to trace using the `-x` option. So if, for example, you want to trace only data flowing over communications networks, use:

```
strmqtrc -x all -t comms
```

For detailed information about the trace command, see strmqtrc.

Selective process tracing

Use the `-p` option of the **`strmqtrc`** command control to restrict trace generation to specified named processes. For example, to trace all threads that result from any running process called amqxxx.exe, use the following command:

```
strmqtrc -p amqxxx.exe
```

For detailed information about the trace command, see strmqtrc.

Related concepts:

"Using trace on UNIX and Linux systems"
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

IBM i "Using trace on IBM MQ server on IBM i" on page 1692
Use the TRCMQM command to start and stop tracing and specify the type of trace that you require.

z/OS "Using trace for problem determination on z/OS" on page 1697
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

"Tracing TLS and SSL: **runmqakm** and iKeyman and iKeycmd functions" on page 1708
How to request **runmqakm** tracing and iKeyman and iKeycmd tracing.

"Tracing additional IBM MQ Java components" on page 1718
For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Using trace on UNIX and Linux systems

Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

UNIX and Linux systems use the following commands for the IBM MQ MQI client trace facility:

strmqtrc
to start tracing

endmqtrc
to end tracing

dspmqtrc <filename>
to display a formatted trace file

The trace facility uses a number of files, which are:

- One file for each entity being traced, in which trace information is recorded
- One additional file on each machine, to provide a reference for the shared memory used to start and end tracing
- One file to identify the semaphore used when updating the shared memory

Files associated with trace are created in a fixed location in the file tree, which is `/var/mqm/trace`.

All client tracing takes place to files in this directory.

You can handle large trace files by mounting a temporary file system over this directory.

On AIX you can use AIX system trace in addition to using the strmqtrc and endmqtrc commands. For more information, see "Tracing with the AIX system trace" on page 1690.

Trace files on IBM MQ for UNIX and Linux systems

Trace files are created in the directory `/var/mqm/trace`.

Note: You can accommodate the production of large trace files by mounting a temporary file system over the directory that contains your trace files. Alternatively, rename the trace directory and create the symbolic link `/var/mqm/trace` to a different directory.

Trace files are named `AMQ` *`ppppp. qq`*`.TRC` where the variables are:

ppppp
: The ID of the process reporting the error.

qq
: A sequence number, starting at 0. If the full file name exists, this value is incremented by one until a unique trace file name is found. A trace file name can exist if a process is reused.

Note:

1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

To format or view a trace file, you must be either the creator of the trace file, or a member of the mqm group.

SSL trace files have the names `AMQ.SSL.TRC` and `AMQ.SSL.TRC.1`. You cannot format SSL trace files; send them unchanged to IBM support.

How to start and stop a trace

In IBM MQ for UNIX and Linux systems, you enable or modify tracing using the **`strmqtrc`** control command (see strmqtrc). To stop tracing, you use the **`endmqtrc`** control command (see endmqtrc). On IBM MQ for Linux (x86 and x86-64 platforms) systems, you can alternatively use the MQ Explorer to start and stop tracing. However, you can trace only everything using the function provided, equivalent to using the commands `strmqtrc -e` and `endmqtrc -e`.

Trace output is unformatted; use the **`dspmqtrc`** control command to format trace output before viewing. For example, to format all trace files in the current directory use the following command:

```
dspmqtrc *.TRC
```

For detailed information about the control command, **`dspmqtrc`**, see dspmqtrc.

Selective component tracing on IBM MQ for UNIX and Linux systems

Use the `-t` and `-x` options to control the amount of trace detail to record. By default, all trace points are enabled. Specify the points you do not want to trace using the `-x` option. If, for example, you want to trace, for queue manager QM1, only output data associated with using Secure Sockets Layer (SSL) channel security, use:

```
strmqtrc -m QM1 -t ssl
```

For detailed information about the trace command, see strmqtrc.

Selective component tracing on IBM MQ for AIX

Use the environment variable MQS_TRACE_OPTIONS to activate the high detail and parameter tracing functions individually.

Because MQS_TRACE_OPTIONS enables tracing to be active without high detail and parameter tracing functions, you can use it to reduce the effect on performance and trace size when you are trying to reproduce a problem with tracing switched on.

Only set the environment variable MQS_TRACE_OPTIONS if you have been instructed to do so by your service personnel.

Typically MQS_TRACE_OPTIONS must be set in the process that starts the queue manager, and before the queue manager is started, or it is not recognized. Set MQS_TRACE_OPTIONS before tracing starts. If it is set after tracing starts it is not recognized.

Selective process tracing on IBM MQ for UNIX and Linux systems

Use the `-p` option of the **strmqtrc** command control to restrict trace generation to specified named processes. For example, to trace all threads that result from any running process called `amqxxx`, use the following command:

```
strmqtrc -p amqxxx
```

For detailed information about the trace command, see strmqtrc.

Related concepts:

IBM i "Using trace on IBM MQ server on IBM i" on page 1692
Use the TRCMQM command to start and stop tracing and specify the type of trace that you require.

z/OS "Using trace for problem determination on z/OS" on page 1697
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

"Tracing TLS and SSL: **runmqakm** and iKeyman and iKeycmd functions" on page 1708
How to request **runmqakm** tracing and iKeyman and iKeycmd tracing.

"Tracing additional IBM MQ Java components" on page 1718
For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Related reference:

"Using trace on Windows" on page 1686
Use the **strmqtrc** and **endmqtrc** commands or the MQ Explorer interface to start and end tracing.

Tracing with the AIX system trace

In addition to the IBM MQ trace, IBM MQ for AIX users can use the standard AIX system trace.

AIX system tracing is a two-step process:

1. Gathering the data
2. Formatting the results

IBM MQ uses two trace hook identifiers:

X'30D' This event is recorded by IBM MQ on entry to or exit from a subroutine.

X'30E' This event is recorded by IBM MQ to trace data such as that being sent or received across a communications network.

Trace provides detailed execution tracing to help you to analyze problems. IBM service support personnel might ask for a problem to be re-created with trace enabled. The files produced by trace can be **very** large so it is important to qualify a trace, where possible. For example, you can optionally qualify a trace by time and by component.

There are two ways to run trace:

1. Interactively.

   The following sequence of commands runs an interactive trace on the program `myprog` and ends the trace.

   ```
   trace -j30D,30E -o trace.file
   ->!myprog
   ->q
   ```

2. Asynchronously.

   The following sequence of commands runs an asynchronous trace on the program `myprog` and ends the trace.

```
trace -a -j30D,30E -o trace.file
myprog
trcstop
```

You can format the trace file with the command:

```
trcrpt -t MQ_INSTALLATION_PATH/lib/amqtrc.fmt trace.file > report.file
```

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

`report.file` is the name of the file where you want to put the formatted trace output.

Note: **All** IBM MQ activity on the machine is traced while the trace is active.

Using trace on HP Integrity NonStop Server

Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file.

Use the following commands on the IBM MQ client for HP Integrity NonStop Server system to use the IBM MQ client trace facility:

strmqtrc
    To start tracing

endmqtrc
    To end tracing

dspmqtrc <filename>
    To display a formatted trace file

The trace facility creates a file for each entity that is being traced. The trace files are created in a fixed location, which is `<mqpath>/var/mqm/trace`. You can handle large trace files by mounting a temporary file system over this directory.

Trace files are named `AMQ.nnn.xx.ppp.qq.TRC` where:

nnn	The name of the process.
xx	The processor number on which the process is running.
ppp	The PIN of the process that you are tracing.
qq	A sequence number, starting at 0. If the full file name exists, this value is incremented by one until a unique trace file name is found. A trace file name can exist if a process is reused.

Note:
1. Each field can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process that is running as part of the entity that is being traced.

Trace files are created in a binary format. To format or view a trace file use the **dspmqtrc** command, you must be either the creator of the trace file, or a member of the mqm group. For example, to format all trace files in the current directory use the following command:

```
dspmqtrc *.TRC
```

For more information about the control command **dspmqtrc**, see dspmqtrc.

How to start and stop a trace

On IBM MQ client for HP Integrity NonStop Server systems, you can enable or modify tracing by using the **strmqtrc** control command, for more information, see strmqtrc. To stop tracing, use the **endmqtrc** control command, for more information, see endmqtrc.

The control commands **strmqtrc** and **endmqtrc** affect tracing only for those processes that are running in one specific processor. By default, this processor is the same as the one in your OSS shell. To enable or end tracing for processes that are running in another processor, you must precede the **strmqtrc** or **endmqtrc** commands with `run -cpu=n` at an OSS shell command prompt, where `n` is the processor number. Here is an example of how to enter the **strmqtrc** command at an OSS shell command prompt:

```
run -cpu=2 strmqtrc
```

This command enables tracing for all processes that are running in processor 2.

The `-m` option to select a queue manager is not relevant for use on the IBM MQ client for HP Integrity NonStop Server . Specifying the `-m` option produces an error.

Use the `-t` and `-x` options to control the amount of trace detail to record. By default, all trace points are enabled. Specify the points that you do not want to trace by using the `-x` option.

Using trace on IBM MQ server on IBM i

IBM i

Use the TRCMQM command to start and stop tracing and specify the type of trace that you require.

There are two stages in using trace:

1. Decide whether you want early tracing. Early tracing lets you trace the creation and startup of queue managers. Note, however, that early trace can easily generate large amounts of trace, because it is implemented by tracing all jobs for all queue managers. To enable early tracing, use TRCMQM with the TRCEARLY parameter set to *YES.
2. Start tracing work using `TRCMQM` **ON*. To stop the trace, you have two options:
   - `TRCMQM` **OFF*, to stop collecting trace records for a queue manager. The trace records are written to files in the `/QIBM/UserData/mqm/trace` directory.
   - `TRCMQM` **END*, to stop collecting trace records for all queue managers and to disable early trace. This option ignores the value of the TRCEARLY parameter.

Specify the level of detail you want, using the TRCLEVEL parameter set to one of the following values:

**DFT*
For minimum-detail level for flow processing trace points.

**DETAIL*
For high-detail level for flow processing trace points.

**PARMS*
For default-detail level for flow processing trace points.

Specify the type of trace output you want, using the OUTPUT parameter set to one of the following values:

***MQM**
Collect binary IBM MQ trace output in the directory specified by the TRCDIR parameter. This value is the default value.

***MQMFMT**
Collect formatted IBM MQ trace output in the directory specified by the TRCDIR parameter.

***PEX**
Collect Performance Explorer (PEX) trace output

***ALL**
Collect both IBM MQ unformatted trace and PEX trace output

Selective trace

You can reduce the amount of trace data being saved, improving runtime performance, using the command `TRCMQM` with `F4=prompt`, then `F9` to customize the TRCTYPE and EXCLUDE parameters:

TRCTYPE
Specifies the type of trace data to store in the trace file. If you omit this parameter, all trace points except those trace points specified in EXCLUDE are enabled.

EXCLUDE
Specifies the type of trace data to omit from the trace file. If you omit this parameter, all trace points specified in TRCTYPE are enabled.

The options available on both TRCTYPE and EXCLUDE are:

***ALL (TRCTYPE only)**
All the trace data as specified by the following keywords is stored in the trace file.

trace-type-list
You can specify more than one option from the following keywords, but each option can occur only once.

***API** Output data for trace points associated with the MQI and major queue manager components.

***CMTRY**
Output data for trace points associated with comments in the IBM MQ components.

***COMMS**
Output data for trace points associated with data flowing over communications networks.

***CSDATA**
Output data for trace points associated with internal data buffers in common services.

***CSFLOW**
Output data for trace points associated with processing flow in common services.

***LQMDATA**
Output data for trace points associated with internal data buffers in the local queue manager.

***LQMFLOW**
Output data for trace points associated with processing flow in the local queue manager.

***OTHDATA**
Output data for trace points associated with internal data buffers in other components.

***OTHFLOW**
Output data for trace points associated with processing flow in other components.

***RMTDATA**
Output data for trace points associated with internal data buffers in the communications component.

***RMTFLOW**
Output data for trace points associated with processing flow in the communications component.

***SVCDATA**
Output data for trace points associated with internal data buffers in the service component.

***SVCFLOW**
Output data for trace points associated with processing flow in the service component.

***VSNDATA**
Output data for trace points associated with the version of IBM MQ running.

Wrapping trace

Use the MAXSTG parameter to wrap trace, and to specify the maximum size of storage to be used for the collected trace records.

The options are:

**DFT*
Trace wrapping is not enabled. For each job, trace data is written to a file with the suffix .TRC until tracing is stopped.

maximum-K-bytes
Trace wrapping is enabled. When the trace file reaches its maximum size, it is renamed with the suffix .TRS, and a new trace file with suffix .TRC is opened. Any existing .TRS file is deleted. Specify a value in the range 1 through 16 000.

Formatting trace output

To format any trace output:

- Enter the QShell
- Enter the command

```
/QSYS.LIB/QMQM.LIB/DSPMQTRC.PGM [-t Format] [-h] [-s]
[-o OutputFileName] InputFileName
```

where:

`InputFileName`
Is a required parameter specifying the name of the file containing the unformatted trace. For example `/QIBM/UserData/mqm/trace/AMQ12345.TRC`.

`-t` *`FormatTemplate`*
Specifies the name of the template file containing details of how to display the trace. The default value is `/QIBM/ProdData/mqm/lib/amqtrc.fmt`.

`-h` Omit header information from the report.

`-s` Extract trace header and put to stdout.

`-o` *`output_filename`*
The name of the file into which to write formatted data.

You can also specify `dspmqtrc *` to format all trace.

Related concepts:

"Using trace on UNIX and Linux systems" on page 1688
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

z/OS "Using trace for problem determination on z/OS" on page 1697
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

"Tracing TLS and SSL: **runmqakm** and iKeyman and iKeycmd functions" on page 1708
How to request **runmqakm** tracing and iKeyman and iKeycmd tracing.

"Tracing additional IBM MQ Java components" on page 1718
For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Related reference:

"Using trace on Windows" on page 1686
Use the **strmqtrc** and **endmqtrc** commands or the MQ Explorer interface to start and end tracing.

Using trace on IBM MQ client on IBM i

IBM i

On IBM i, there is no Control Language (CL) command to capture the trace when using a standalone IBM MQ MQI client. STRMQTRC and ENDMQTRC programs can be used to enable and disable the trace.

Example for start trace:

```
CALL PGM(QMQM/STRMQTRC) PARM('-e' '-t' 'all' '-t' 'detail')
Where -e option requests early tracing of all the process -t option for trace type
```

To end the trace

```
CALL PGM(QMQM/ENDMQTRC) PARM('-e')
```

- Optional parameters:

  -t *TraceType*

  The points to trace and the amount of trace detail to record. By default all trace points are enabled and a default-detail trace is generated.

  Alternatively, you can supply one or more of the options in Table 1. For each *TraceType* value you specify, including -t all, specify either -t parms or -t detail to obtain the appropriate level of trace detail. If you do not specify either -t parms or -t detail for any particular trace type, only a default-detail trace is generated for that trace type.

  If you supply multiple trace types, each must have its own -t flag. You can include any number of -t flags, if each has a valid trace type associated with it.

  It is not an error to specify the same trace type on multiple -t flags.

  See the following table for allowed values for *TraceType*.

Table 159. TraceType values

Value	**Description**
all	Output data for every trace point in the system (the default). Using *all* activates tracing at default detail level.
api	Output data for trace points associated with the message queue interface (MQI) and major queue manager components.
commentary	Output data for trace points associated with comments in the IBM MQ components.
comms	Output data for trace points associated with data flowing over communications networks.
csdata	Output data for trace points associated with internal data buffers in common services.
csflows	Output data for trace points associated with processing flow in common services.
detail	Activate tracing at high-detail level for flow processing trace points.
lqmdata	Output data for trace points associated with internal data buffers in the local queue manager.
lqmflows	Output data for trace points associated with processing flow in the local queue manager.
otherdata	Output data for trace points associated with internal data buffers in other components.
otherflows	Output data for trace points associated with processing flow in other components.
parms	Activate tracing at default-detail level for flow processing trace points.
remotedata	Output data for trace points associated with internal data buffers in the communications component.
remoteflows	Output data for trace points associated with processing flow in the communications component.
servicedata	Output data for trace points associated with internal data buffers in the service component.
serviceflows	Output data for trace points associated with processing flow in the service component.
versiondata	Output data for trace points associated with the version of IBM MQ running.

-x *TraceType*
: The points not to trace. By default all trace points are enabled and a default-detail trace is generated. The *TraceType* values you can specify are the same as the values listed for the -t flag in Table 1.

  You can use the -x flag with *TraceType* values to exclude those trace points you do not want to record. Excluding specified trace points is useful in reducing the amount of trace produced.

  If you supply multiple trace types, each must have its own -x flag. You can include any number of -x flags, if each has a valid *TraceType* associated with it.

-s Reports the tracing options that are currently in effect. You must use this parameter on its own with no other parameters.

A limited number of slots are available for storing trace commands. When all slots are in use, then no more trace commands can be accepted unless they replace an existing slot. Slot numbers are not fixed, so if the command in slot number 0 is removed, for example by an **endmqtrc** command, then all the other slots move up, with slot 1 becoming slot 0, for example. An asterisk (*) in a field means that no value is defined, and is equivalent to the asterisk wildcard.

-l *MaxSize*
The maximum size of a trace file (`AMQppppp.qq.TRC`) in megabytes (MB). For example, if you specify a *MaxSize* of 1, the size of the trace is limited to 1 MB.

When a trace file reaches the specified maximum, it is renamed to `AMQppppp.qq.TRS` and a new `AMQppppp.qq.TRC` file is started. If a previous copy of an `AMQppppp.qq.TRS` file exists, it is deleted.

The highest value that *MaxSize* can be is 2048 MB.

-e Requests early tracing of all processes

For more details see the **strmqtrc** command

- To end the trace:

```
/QSYS.LIB/QMQM.LIB/ENDMQTRC.PGM [-e] [-a]
```

where:

-e Ends early tracing of all processes.

Using **endmqtrc** with no parameters has the same effect as **endmqtrc -e**. You cannot specify the -e flag with the -m flag, the -i flag, or the -p flag.

-a Ends all tracing.

For more details see the endmqtrc **endmqtrc** command

- To display a formatted trace file:

```
/QSYS.LIB/QMQM.LIB/DSPMQTRC.pgm
```

To examine FFST files, see the First Failure Support Technology (FFST) for IBM MQ for IBM i.

Related concepts:

"Using trace on UNIX and Linux systems" on page 1688
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

z/OS "Using trace for problem determination on z/OS"
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

"Tracing TLS and SSL: **runmqakm** and iKeyman and iKeycmd functions" on page 1708
How to request **runmqakm** tracing and iKeyman and iKeycmd tracing.

"Tracing additional IBM MQ Java components" on page 1718
For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Related reference:

"Using trace on Windows" on page 1686
Use the **strmqtrc** and **endmqtrc** commands or the MQ Explorer interface to start and end tracing.

Using trace for problem determination on z/OS

z/OS

There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

The trace facilities available with IBM MQ for z/OS are:
- The user parameter (or API) trace
- The IBM internal trace used by the support center
- The channel initiator trace
- The line trace

Use the following links to find out how to collect and interpret the data produced by the user parameter trace, and describes how to produce the IBM internal trace for use by the IBM support center. There is also information about the other trace facilities that you can use with IBM MQ.
- Controlling the GTF for your z/OS system
- Controlling the IBM MQ trace for each queue manager subsystem for which you want to collect data
- "Formatting and identifying the control block information" on page 1700
- "Interpreting the trace information" on page 1701

If trace data is not produced, check the following:
- Was the GTF started correctly, specifying EIDs 5E9, 5EA, and 5EE on the USRP option?
- Was the START TRACE(GLOBAL) command entered correctly, and were the relevant classes specified?

For more information about other trace options available on z/OS, see "Other types of trace" on page 1703.

Related concepts:

"Using trace on UNIX and Linux systems" on page 1688
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

"Using trace on IBM MQ server on IBM i" on page 1692
Use the TRCMQM command to start and stop tracing and specify the type of trace that you require.

"Tracing TLS and SSL: **runmqakm** and iKeyman and iKeycmd functions" on page 1708
How to request **runmqakm** tracing and iKeyman and iKeycmd tracing.

"Tracing additional IBM MQ Java components" on page 1718
For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Related reference:

"Using trace on Windows" on page 1686
Use the **strmqtrc** and **endmqtrc** commands or the MQ Explorer interface to start and end tracing.

The MQI call and user parameter, and z/OS generalized trace facility (GTF)

Use this topic to understand how to control GTF and IBM MQ trace.

You can obtain information about MQI calls and user parameters passed by some IBM MQ calls on entry to, and exit from, IBM MQ. To do this, use the global trace in conjunction with the z/OS generalized trace facility (GTF).

Controlling the GTF:

Use this topic to understand how to start and stop the GTF.

- Starting the GTF
- Stopping the GTF

Starting the GTF

When you start the GTF, specify the USRP option. You are prompted to enter a list of event identifiers (EIDs). The EIDs used by IBM MQ are:

5E9	To collect information about control blocks on entry to IBM MQ
5EA	To collect information about control blocks on exit from IBM MQ

Sometimes, if an error occurs that you cannot solve yourself, you might be asked by your IBM support center to supply other, internal, trace information for them to analyze. The additional type of trace is:

5EE	To collect information internal to IBM MQ

You can also use the JOBNAMEP option, specifying the batch, CICS, IMS, or TSO job name, to limit the trace output to specific jobs. Figure 123 on page 1699 illustrates sample startup for the GTF, specifying the four EIDs, and a jobname. The lines shown in bold type **like this** are the commands that you enter at the console; the other lines are prompts and responses.

For more information about starting the GTF trace, see the *MVS Diagnosis: Tools and Service Aids* manual.

```
START GTFxx.yy
 #HASP100 GTFxx.yy ON STCINRDR
 #HASP373 GTFxx.yy STARTED
*01 AHL100A SPECIFY TRACE OPTIONS
 R 01,TRACE=JOBNAMEP,USRP
 TRACE=JOBNAMEP,USRP
 IEE600I REPLY TO 01 IS;TRACE=JOBNAMEP,USRP
*02 ALH101A SPECIFY TRACE EVENT KEYWORDS - JOBNAME=,USR=
 R 02,JOBNAME=(xxxxMSTR,xxxxCHIN,zzzzzzzz),USR=(5E9,5EA,5EE)
 JOBNAME=(xxxxMSTR,xxxxCHIN,zzzzzzzz),USR=(5E9,5EA,5EE)
 IEE600I REPLY TO 02 IS;JOBNAME=(xxxxMSTR,xxxxCHIN,zzzzzzzz),USR=(5E9,5EA,5EE)
*03 ALH102A CONTINUE TRACE DEFINITION OR REPLY END
 R 03,END
 END
 IEE600I REPLY TO 03 IS;END
 AHL103I TRACE OPTIONS SELECTED-USR=(5E9,5EA,5EE)
 AHL103I JOBNAME=(xxxxMSTR,xxxxCHIN,zzzzzzzz)
*04 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
 R 04,U
 U
 IEE600I REPLY TO 04 IS;U
 AHL031I GTF INITIALIZATION COMPLETE

where:

     xx is the name of the GTF procedure to use (optional), and yy is an
     identifier for this occurrence of GTF trace.

     xxxx is the name of the queue manager and zzzzzzzz is
     a batch job or CICS region name. Up to 5 job names can be listed.
```

Figure 123. Example startup of GTF to use with the IBM MQ trace

When using GTF, specify the primary job name (CHINIT, CICS, or batch) in addition to the queue manager name (xxxxMSTR).

Stopping the GTF

When you stop the GTF, you must specify the additional identifier (**yy**) used at startup. Figure 124 illustrates a sample stop command for the GTF. The commands shown in bold type **like this** are the commands that you enter at the console.

```
 STOP yy
```

Figure 124. Example of GTF Stop command to use with the IBM MQ trace

Related information:

Generating IBM MQ GTF trace on IBM z/OS

Controlling the trace within IBM MQ:

IBM MQ for z/OS trace is controlled using MQSC commands. Use this topic to understand how to control the trace, and the type of trace information that is output.

Use the START TRACE command, specifying type GLOBAL to start writing IBM MQ records to the GTF. You must also specify dest(GTF), for example in the following command:

```
/cpf start trace(G)class(2,3)dest(GTF)
```

To define the events that you want to produce trace data for, use one or more of the following classes:

CLASS	Event traced
2	Record the MQI call and MQI parameters when a completion code other than MQRC_NONE is detected.
3	Record the MQI call and MQI parameters on entry to and exit from the queue manager.

After the trace has started, you can display information about, alter the properties of, and stop, the trace with the following commands:
- DISPLAY TRACE
- ALTER TRACE
- STOP TRACE

To use any of the trace commands, you must have one of the following:
- Authority to issue start and stop trace commands (trace authority)
- Authority to issue the display trace command (display authority)

Note:
1. The trace commands can also be entered through the initialization input data sets.
2. The trace information produced will also include details of syncpoint flows - for example PREPARE and COMMIT.

For information about these commands, see MQSC commands.

Formatting and identifying the control block information:

After capturing a trace, the output must be formatted and the IBM MQ control blocks identified.
- Formatting the information
- Identifying the control blocks associated with IBM MQ
- Identifying the event identifier associated with the control block

Formatting the information

To format the user parameter data collected by the global trace, use either the batch job shown in Figure 125 on page 1701 or the IPCS `GTFTRACE USR(` *xxx* `)` command, where *xxx* is:

5E9	To format information about control blocks on entry to IBM MQ MQI calls
5EA	To format information about control blocks on exit from IBM MQ MQI calls
5EE	To format information about IBM MQ internals

You can also specify the JOBNAME(*jobname*) parameter to limit the formatted output to specific jobs.

```
//S1 EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=4096K
//IPCSPARM DD DSN=SYS1.PARMLIB,DISP=SHR
//IPCSDDIR DD DSN=thlqual.ipcs.dataset.directory,DISP=SHR
//SYSTSPRT DD SYSOUT=*,DCB=(LRECL=137)
//IPCSTOC  DD SYSOUT=*
//GTFIN    DD DSN=gtf.trace,DISP=SHR
//SYSTSIN  DD *
IPCS
SETDEF FILE(GTFIN) NOCONFIRM
GTFTRACE USR(5E9,5EA,5EE)
/*
//STEPLIB  DD  DSN=thlqual.SCSQAUTH,DISP=SHR
```

Figure 125. Formatting the GTF output in batch. thlqual is your high level qualifier for IBM MQ data sets, and gtf.trace is the name of the data set containing your trace information. You must also specify your IPCS data set directory.

Identifying the control blocks associated with IBM MQ

The format identifier for the IBM MQ trace is D9. This value appears at the beginning of each formatted control block in the formatted GTF output, in the form:

```
USRD9
```

Identifying the event identifier associated with the control block

The trace formatter inserts one of the following messages at the top of each control block. These indicate whether the data was captured on entry to or exit from IBM MQ

- CSQW072I ENTRY: MQ user parameter trace
- CSQW073I EXIT: MQ user parameter trace

Related concepts:

"Controlling the GTF" on page 1698
Use this topic to understand how to start and stop the GTF.

Interpreting the trace information:

The GTFTRACE produced by IBM MQ can be examined to determine possible errors with invalid addresses, invalid control blocks, and invalid data.

When you look at the data produced by the GTFTRACE command, consider the following points:

- If the control block consists completely of zeros, it is possible that an error occurred while copying data from the user's address space. This might be because an invalid address was passed.
- If the first part of the control block contains non-null data, but the rest consists of zeros, it is again possible that an error occurred while copying data from the user's address space, for example, the control block was not placed entirely within valid storage. This might also be due to the control block not being initialized correctly.
- If the error occurred on exit from IBM MQ, it is possible that IBM MQ might not write the data to the user's address space. The data displayed is the version that it was attempting to copy to the user's address space.

The following tables show details of the control blocks that are traced.

Table 160 on page 1702 illustrates which control blocks are traced for different MQI calls.

Table 160. Control blocks traced for IBM MQ MQI calls

MQI call	Entry	Exit
MQCB	MQCBD, MQMD, MQGMO	MQCBD, MQMD, MQGMO
MQCLOSE	None	None
MQGET	MQMD, MQGMO	MQMD, MQGMO, and the first 256 bytes of message data
MQINQ	Selectors (if *SelectorCount* is greater than 0)	Selectors (if *SelectorCount* is greater than 0) Integer attributes (if *IntAttrCount* is greater than 0) Character attributes (if *CharAttrLength* is greater than 0)
MQOPEN	MQOD	MQOD
MQPUT	MQMD, MQPMO, and the first 256 bytes of message data	MQMD, MQPMO, and the first 256 bytes of message data
MQPUT1	MQMD, MQOD, MQPMO, and the first 256 bytes of message data	MQMD, MQOD, MQPMO, and the first 256 bytes of message data
MQSET	Selectors (if *SelectorCount* is greater than 0) Integer attributes (if *IntAttrCount* is greater than 0) Character attributes (if *CharAttrLength* is greater than 0)	Selectors (if *SelectorCount* is greater than 0) Integer attributes (if *IntAttrCount* is greater than 0) Character attributes (if *CharAttrLength* is greater than 0)
MQSTAT	MQSTS	MQSTS
MQSUB	MQSD, MQSD.ObjectString, MQSD.SubName, MQSD.SubUserData, MQSD.SelectionString, MQSD.ResObjectString	MQSD, MQSD.ObjectString, MQSD.SubName, MQSD.SubUserData, MQSD.SelectionString, MQSD.ResObjectString
MQSUBRQ	MQSRO	MQSRO

Note: In the special case of an **MQGET** call with the WAIT option, a double entry is seen if there is no message available at the time of the **MQGET** request, but a message subsequently becomes available before the expiry of any time interval specified.

This is because, although the application has issued a single **MQGET** call, the adapter is performing the wait on behalf of the application and when a message becomes available it reissues the call. So in the trace it appears as a second **MQGET** call.

Information about specific fields of the queue request parameter list is also produced in some circumstances. The fields in this list are identified as follows:

Identifier	Description
Action	Requested action
BufferL	Buffer length
CBD	Address of callback descriptor
CompCode	Completion code
CharAttL	Character attributes length
DataL	Data length
Hobj	Object handle

Identifier	Description
Hsub	Subscription handle
IntAttC	Count of integer attributes
pObjDesc	Object descriptor
Oper	Operation
Options	Options
pBuffer	Address of buffer
pCharAtt	Address of character attributes
pCTLO	Address of control callback options
pECB	Address of ECB used in get
pGMO	Address of get message options
pIntAtt	Address of integer attributes
pMsgDesc	Address of message descriptor
pPMO	Address of put message options
pSD	Address of subscription descriptor
pSelect	Address of selectors
pSRQOpt	Address of subscription request options
pSTS	Address of status structure
Reason	Reason code
RSVn	Reserved for IBM
SelectC	Selector count
Thread	Thread
Type	Requested type
UOWInfo	Information about the unit of work
Userid	CICS or IMS user ID, for batch or TSO this is zero

Other types of trace

There are other trace facilities available for problem determination. Use this topic to investigate channel initiator trace, line trace, CICS adapter trace, SSL trace, and z/OS trace.

It can be helpful to use the following trace facilities with IBM MQ.

- The channel initiator trace
- The line trace
- The CICS adapter trace
- System SSL trace
- z/OS z/OS traces

The channel initiator trace

See Figure 130 on page 1737 for information about how to get a dump of the channel initiator address space. Note that dumps produced by the channel initiator do not include trace data space. The trace data space, which is called CSQXTRDS, contains trace information. You can request this by specifying it on a slip trap or when you use the dump command.

You can run the trace using the START TRACE command. You can also set this trace to start automatically using the TRAXSTR queue manager attribute. For more information about how to do this, see ALTER QMGR.

You can display this trace information by entering the IPCS command:

```
LIST 1000. DSPNAME(CSQXTRDS)
```

You can format it using the command:

```
CTRACE COMP(CSQXssnm)
```

where *ssnm* is the subsystem name.

The line trace

A wrap-around line trace exists for each channel. This trace is kept in a 4 KB buffer for each channel in the channel initiator address space. Trace is produced for each channel, so it is ideal for problems where a channel appears to be hung, because information can be collected about the activity of this channel long after the normal trace has wrapped.

The line trace is always active; you cannot turn it off. It is available for both LU 6.2 and TCP channels and should reduce the number of times a communications trace is required.

You can view the trace as unformatted trace that is written to CSQSNAP. You can display the trace by following these steps:

1. Ensure that the CHIN procedure has a SNAP DD statement.
2. Start a CHIN trace, specifying IFCID 202 as follows:
   ```
   START TRACE(CHINIT) CLASS(4) IFCID(202)
   ```
3. Display the channel status for those channels for which the line trace is required:
   ```
   DISPLAY CHSTATUS(channel) SAVED
   ```

   This dumps the current line for the selected channels to CSQSNAP. See "Snap dumps" on page 1753 for further information.

   Note:

   a. The addresses of the storage dump are incorrect because the CSQXFFST mechanism takes a copy of the storage before writing it to CSQSNAP.
   b. The dump to CSQSNAP is only produced the first time you run the DISPLAY CHSTATUS SAVED command. This is to prevent getting dumps each time you run the command.

      To obtain another dump of line trace data, you must stop and restart the current trace.

      1) You can use a selective STOP TRACE command to stop just the trace that was started to gather the line trace data. To do this, note the TRACE NUMBER assigned to the trace as shown in this example:
         ```
         +ssid START TRACE(CHINIT) CLASS(4) IFCID(202)
              CSQW130I +ssid 'CHINIT' TRACE STARTED, ASSIGNED TRACE NUMBER 01
         ```
      2) To stop the trace, issue the following command:
         ```
         +ssid STOP TRACE(CHINIT) TNO(01)
         ```
      3) You can then enter another START TRACE command with a DISPLAY CHSTATUS SAVED command to gather more line trace data to CSQSNAP.
4. The line trace buffer is unformatted. Each entry starts with a clock, followed by a time stamp, and an indication of whether this is an OUTBOUND or INBOUND flow. Use the time stamp information to find the earliest entry.

The CICS adapter trace

The CICS adapter writes entries to the CICS trace if your trace number is set to a value in the range 0 through 199 (decimal), and if either:
- CICS user tracing is enabled, or
- CICS internal/auxiliary trace is enabled

You can enable CICS tracing in one of two ways:
- Dynamically, using the CICS-supplied transaction CETR
- By ensuring that the USERTR parameter in the CICS system initialization table (SIT) is set to YES

For more information about enabling CICS trace, see the *CICS Problem Determination Guide*.

The CICS trace entry originating from the CICS adapter has a value AP0 *000*, where *000* is the hexadecimal equivalent of the decimal value of the CICS adapter trace number you specified.

The trace entries are shown in "CICS adapter trace entries."

System SSL trace

You can collect System SSL trace using the SSL Started Task. The details of how to set up this task are in the *System Secure Sockets Layer Programming* documentation, SC24-5901. A trace file is generated for each SSLTASK running in the CHINIT address space.

z/OS

z/OS traces

z/OS traces, which are common to all products operating as formal subsystems of z/OS, are available for use with IBM MQ. For information about using and interpreting this trace facility, see the *MVS Diagnosis: Tools and Service Aids* manual.

CICS adapter trace entries:

Use this topic as a reference for CICS adapter trace entries.

The CICS trace entry for these values is AP0 xxx (where xxx is the hexadecimal equivalent of the trace number you specified when the CICS adapter was enabled). These trace entries are all issued by CSQCTRUE, except CSQCTEST, which is issued by CSQCRST and CSQCDSP.

Table 161. CICS adapter trace entries

Name	Description	Trace sequence	Trace data
CSQCABNT	Abnormal termination	Before issuing END_THREAD ABNORMAL to IBM MQ. This is due to the end of the task and therefore an implicit backout could be performed by the application. A ROLLBACK request is included in the END_THREAD call in this case.	Unit of work information. You can use this information when finding out about the status of work. (For example, it can be verified against the output produced by the DISPLAY THREAD command, or the log print utility.)
CSQCAUID	Bridge security	Before validating bridge user password or PassTicket.	User ID.
CSQCBACK	Syncpoint backout	Before issuing BACKOUT to IBM MQ. This is due to an explicit backout request from the application.	Unit of work information.

Table 161. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCCONX	MQCONNX	Before issuing **MQCONNX** to IBM MQ.	Connection tag.
CSQCCCRC	Completion code and reason code	After unsuccessful return from API call.	Completion code and reason code.
CSQCCOMM	Syncpoint commit	Before issuing COMMIT to IBM MQ. This can be due to a single-phase commit request or the second phase of a two-phase commit request. The request is due to a explicit syncpoint request from the application.	Unit of work information.
CSQCDCFF	IBM use only		
CSQCDCIN	IBM use only		
CSQCDCOT	IBM use only		
CSQCEXER	Execute resolve	Before issuing EXECUTE_RESOLVE to IBM MQ.	The unit of work information of the unit of work issuing the EXECUTE_RESOLVE. This is the last in-doubt unit of work in the resynchronization process.
CSQCGETW	GET wait	Before issuing CICS wait.	Address of the ECB to be waited on.
CSQCGMGD	GET message data	After successful return from **MQGET** .	Up to 40 bytes of the message data.
CSQCGMGH	GET message handle	Before issuing **MQGET** to IBM MQ.	Object handle.
CSQCGMGI	Get message ID	After successful return from **MQGET** .	Message ID and correlation ID of the message.
CSQCHCER	Hconn error	Before issuing any MQ verb.	Connection handle.
CSQCINDL	In-doubt list	After successful return from the second INQUIRE_INDOUBT.	The in-doubt units of work list.
CSQCINDO	IBM use only		
CSQCINDS	In-doubt list size	After successful return from the first INQUIRE_INDOUBT and the in-doubt list is not empty.	Length of the list; divided by 64 gives the number of in-doubt units of work.
CSQCINDW	Syncpoint in doubt	During syncpoint processing, CICS is in doubt as to the disposition of the unit of work.	Unit of work information.
CSQCINQH	INQ handle	Before issuing **MQINQ** to IBM MQ.	Object handle.
CSQCLOSH	CLOSE handle	Before issuing **MQCLOSE** to IBM MQ.	Object handle.
CSQCLOST	Disposition lost	During the resynchronization process, CICS informs the adapter that it has been cold started so no disposition information regarding the unit of work being resynchronized is available.	Unit of work ID known to CICS for the unit of work being resynchronized.

Table 161. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCNIND	Disposition not in doubt	During the resynchronization process, CICS informs the adapter that the unit of work being resynchronized should not have been in doubt (that is, perhaps it is still running).	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCNORT	Normal termination	Before issuing END_THREAD NORMAL to IBM MQ. This is due to the end of the task and therefore an implicit syncpoint commit might be performed by the application. A COMMIT request is included in the END_THREAD call in this case.	Unit of work information.
CSQCOPNH	OPEN handle	After successful return from **MQOPEN** .	Object handle.
CSQCOPNO	OPEN object	Before issuing **MQOPEN** to IBM MQ.	Object name.
CSQCPMGD	PUT message data	Before issuing **MQPUT** to IBM MQ.	Up to 40 bytes of the message data.
CSQCPMGH	PUT message handle	Before issuing **MQPUT** to IBM MQ.	Object handle.
CSQCPMGI	PUT message ID	After successful **MQPUT** from IBM MQ.	Message ID and correlation ID of the message.
CSQCPREP	Syncpoint prepare	Before issuing PREPARE to IBM MQ in the first phase of two-phase commit processing. This call can also be issued from the distributed queuing component as an API call.	Unit of work information.
CSQCP1MD	PUTONE message data	Before issuing **MQPUT1** to IBM MQ.	Up to 40 bytes of data of the message.
CSQCP1MI	PUTONE message ID	After successful return from **MQPUT1** .	Message ID and correlation ID of the message.
CSQCP1ON	PUTONE object name	Before issuing **MQPUT1** to IBM MQ.	Object name.
CSQCRBAK	Resolved backout	Before issuing RESOLVE_ROLLBACK to IBM MQ.	Unit of work information.
CSQCRCMT	Resolved commit	Before issuing RESOLVE_COMMIT to IBM MQ.	Unit of work information.
CSQCRMIR	RMI response	Before returning to the CICS RMI (resource manager interface) from a specific invocation.	Architected RMI response value. Its meaning depends of the type of the invocation. To determine the type of invocation, look at previous trace entries produced by the CICS RMI component.
CSQCRSYN	Resync	Before the resynchronization process starts for the task.	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCSETH	SET handle	Before issuing **MQSET** to IBM MQ.	Object handle.
CSQCTASE	IBM use only		

Table 161. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCTEST	Trace test	Used in EXEC CICS ENTER TRACE call to verify the trace number supplied by the user or the trace status of the connection.	No data.

Tracing TLS and SSL: `runmqakm` and iKeyman and iKeycmd functions

How to request **`runmqakm`** tracing and iKeyman and iKeycmd tracing.

iKeyman and iKeycmd trace

To request iKeyman tracing, execute the iKeyman command for your platform with the following -D flags.

For Windows UNIX and Linux systems:

```
strmqikm -Dkeyman.debug=true -Dkeyman.jnitracing=ON
```

To request iKeycmd tracing, run the iKeycmd command for your platform with the following -D flags.

For Windows UNIX and Linux systems:

```
runmqckm -Dkeyman.debug=true -Dkeyman.jnitracing=ON
```

iKeyman and iKeycmd write three trace files to the directory from which you start them, so consider starting iKeyman or iKeycmd from the trace directory to which the runtime TLS or SSL trace is written: `/var/mqm/trace` on UNIX and Linux systems and *`MQ_INSTALLATION_PATH`*`/trace` on Windows. *`MQ_INSTALLATION_PATH`* represents the high-level directory in which IBM MQ is installed.

The trace file generated by iKeyman and iKeycmd has the following format:

```
debugTrace. n
```

where *n* is an incrementing number starting at 0.

`runmqakm` trace

To request **`runmqakm`** tracing, execute the **`runmqakm`** command with the following flags:

```
runmqakm -trace filename
```

where *filename* is the name of the trace file to create. You cannot format the **`runmqakm`** trace file. Send it unchanged to IBM support. The **`runmqakm`** trace file is a binary file and, if it is transferred to IBM support via FTP, it must be transferred in binary transfer mode.

Runtime TLS and SSL trace

On UNIX, Linux, and Windows systems, you can independently request trace information for iKeyman, iKeycmd, the runtime TLS or SSL functions, or a combination of these.

The runtime TLS trace files have the names AMQ.TLS.TRC and AMQ.TLS.TRC.1 and the SSL trace files have the names AMQ.SSL.TRC and AMQ.SSL.TRC.1. You cannot format any of the TLS or SSL trace files; send them unchanged to IBM support. The TLS and SSL trace files are binary files and, if they are transferred to IBM support via FTP, they must be transferred in binary transfer mode.

Related concepts:

"Using trace on UNIX and Linux systems" on page 1688
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

IBM i "Using trace on IBM MQ server on IBM i" on page 1692
Use the TRCMQM command to start and stop tracing and specify the type of trace that you require.

z/OS "Using trace for problem determination on z/OS" on page 1697
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

"Tracing additional IBM MQ Java components" on page 1718
For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Related reference:

"Using trace on Windows" on page 1686
Use the **strmqtrc** and **endmqtrc** commands or the MQ Explorer interface to start and end tracing.

Tracing IBM MQ classes for JMS applications

The trace facility in IBM MQ classes for JMS is provided to help IBM Support to diagnose customer issues. Various properties control the behavior of this facility.

If you are asked to provide trace output to investigate an issue, use one of the options mentioned below:

- If the issue is easy to recreate, then collect an IBM MQ classes for JMS trace by using a Java System Property. For more information, see "Collecting an IBM MQ classes for JMS trace by using a Java system property" on page 1710.
- If an application needs to run for a period of time before the issue occurs, collect an IBM MQ classes for JMS trace by using the IBM MQ classes for JMS configuration file. For more information, see "Collecting an IBM MQ classes for JMS trace by using the IBM MQ classes for JMS configuration file" on page 1711.
- To generate a trace from an application that is currently running, collect the IBM MQ classes for JMS trace dynamically by using the traceControl utility. For more information, see "Collecting an IBM MQ classes for JMS trace dynamically by using the traceControl utility" on page 1712.

If you are unsure which option to use, contact your IBM Support representative and they will be able to advise you on the best way to collect trace for the issue that you are seeing.

If a severe or unrecoverable error occurs, First Failure Support Technology (FFST) information is recorded in a file with a name of the format `JMSCC` *`xxxx`*`.FDC` where *`xxxx`* is a four-digit number. This number is incremented to differentiate `.FDC` files.

`.FDC` files are always written to a subdirectory called `FFDC`. The subdirectory is in one of two locations, depending on whether trace is active:

Trace is active, and *`traceOutputName`* is set
The FFDC directory is created as a subdirectory of the directory to which the trace file is being written.

Trace is not active or *`traceOutputName`* is not set
The FFDC directory is created as a subdirectory of the current working directory.

For more information about FFST in IBM MQ classes for JMS, see "FFST: IBM MQ classes for JMS" on page 1667.

The JSE common services uses `java.util.logging` as its trace and logging infrastructure. The root object of this infrastructure is the LogManager. The log manager has a reset method that closes all handlers and sets the log level to `null`, which in effect turns off all the trace. If your application or application server calls java.util.logging.LogManager.getLogManager().reset(), it closes all trace, which might prevent you from diagnosing any problems. To avoid closing all trace, create a LogManager class with an overridden reset() method that does nothing, as shown in the following example:

```
package com.ibm.javaut.tests;
import java.util.logging.LogManager;
public class JmsLogManager extends LogManager {
 // final shutdown hook to ensure that the trace is finally shutdown
 // and that the lock file is cleaned-up
 public class ShutdownHook extends Thread{
  public void run(){
   doReset();
  }
 }
  public JmsLogManager(){
  // add shutdown hook to ensure final cleanup
  Runtime.getRuntime().addShutdownHook(new ShutdownHook());
 }
  public void reset() throws SecurityException {
  // does nothing
 }
 public void doReset(){
  super.reset();
 }
        }
```

The shutdown hook is necessary to ensure that trace is properly shut down when the JVM finishes. To use the modified log manager instead of the default one, add a system property to the JVM startup:

```
java -Djava.util.logging.manager=com. mycompany.logging.LogManager ...
```

Collecting an IBM MQ classes for JMS trace by using a Java system property

For issues that can be reproduced in a short amount of time, IBM MQ classes for JMS trace should be collected by setting a Java system property when starting the application.

About this task

To collect a trace by using a Java system property, complete the following steps.

Procedure

Run the application that is going to be traced by using the following command:

```
java -Dcom.ibm.msg.client.commonservices.trace.status=ON application_name
```

When the application starts, the IBM MQ classes for JMS start writing trace information to a trace file in the application's current working directory. The name of the trace file depends on the environment that the application is running in:

- For IBM MQ classes for JMS for Version 8.0.0, Fix Pack 6 or earlier, trace is written to a file called `mqjms_%PID%.trc`.
- V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for JMS from the JAR file `com.ibm.mqjms.jar`, trace is written to a file called `mqjms_%PID%.trc`.
- V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for JMS from the relocatable JAR file `com.ibm.mq.allclient.jar`, trace is written to a file called `mqjavaclient_%PID%.trc`.

where *%PID%* is the process identifier of the application that is being traced.
The application stops writing information to the trace file when it is stopped.
If the application has to run for a long period of time before the issue that the trace is being collected for occurs, then the trace file could potentially be very large. In this situation, consider collecting trace by

using the IBM MQ classes for JMS configuration file (see "Collecting an IBM MQ classes for JMS trace by using the IBM MQ classes for JMS configuration file"). When enabling trace in this way, it is possible to control the amount of trace data that the IBM MQ classes for JMS generate.

Collecting an IBM MQ classes for JMS trace by using the IBM MQ classes for JMS configuration file

If an application must run for a long period of time before an issue occurs, IBM MQ classes for JMS trace should be collected by using the IBM MQ classes for JMS configuration file. The configuration file allows you to specify various options to control the amount of trace data that is collected.

About this task

To collect a trace by using the IBM MQ classes for JMS configuration file, complete the following steps.

Procedure

1. Create an IBM MQ classes for JMS configuration file. For more information about this file, see The IBM MQ classes for JMS configuration file.
2. Edit the IBM MQ classes for JMS configuration file so that the property **`com.ibm.msg.client.commonservices.trace.status`** is set to the value `ON`.
3. Optional: Edit the other properties that are listed in the IBM MQ classes for JMS configuration file Java Standard Edition Trace Settings.
4. Run the IBM MQ classes for JMS application by using the following command:

   ```
   java -Dcom.ibm.msg.client.config.location=config_file_url
   application_name
   ```

   where *config_file_url* is a uniform resource locator (URL) that specifies the name and location of the IBM MQ classes for JMS configuration file. URLs of the following types are supported: `http`, `file`, `ftp`, and `jar`.

   Here is an example of a Java command:

   ```
   java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config
   MyAppClass
   ```

   This command identifies the IBM MQ classes for JMS configuration file as the file `D:\mydir\myjms.config` on the local Windows system.

   By default, the IBM MQ classes for JMS start writing trace information to a trace file in the application's current working directory when the application starts up. The name of the trace file depends on the environment that the application is running in:

   - For IBM MQ classes for JMS for Version 8.0.0, Fix Pack 6 or earlier, trace is written to a file called `mqjms_%PID%.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for JMS from the JAR file `com.ibm.mqjms.jar`, trace is written to a file called `mqjms_%PID%.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for JMS from the relocatable JAR file `com.ibm.mq.allclient.jar`, trace is written to a file called `mqjavaclient_%PID%.trc`.

   where *%PID%* is the process identifier of the application that is being traced.

   To change the name of the trace file, and the location where it is written, ensure that the IBM MQ classes for JMS configuration file that the application uses contains an entry for the property **`com.ibm.msg.client.commonservices.trace.outputName`**. The value for the property can be either of the following:

   - The name of the trace file that is created in the application's working directory.
   - The fully qualified name of the trace file, including the directory in which the file is created.

For example, to configure the IBM MQ classes for JMS to write trace information for an application to a file called `C:\Trace\trace.trc`, the IBM MQ classes for JMS configuration file that the application uses needs to contain the following entry:

```
com.ibm.msg.client.commonservices.trace.outputName=C:\Trace\trace.trc
```

Collecting an IBM MQ classes for JMS trace dynamically by using the traceControl utility

The traceControl utility that is shipped with the IBM MQ classes for JMS allows trace to be collected from a running application. This can be very useful if IBM Support need to see a trace from an application once an issue has occurred, or if trace needs to be collected from a critical application that cannot be stopped.

About this task

For more information about the traceControl utility, see "Controlling trace in a running process by using IBM MQ classes for Java and IBM MQ classes for JMS" on page 1721.

To collect a trace by using the traceControl utility, complete the following steps.

Procedure

1. Bring up a command prompt, and navigate to the directory *MQ_INSTALLATION_PATH*`\java\lib`.
2. Run the command:
   ```
   java -jar com.ibm.mq.traceControl -list
   ```
   This command brings up a list of all of the Java processes on the system.
3. Identify the process identifier for the IBM MQ classes for JMS application that needs to be traced, and run the command:
   ```
   java -jar com.ibm.mq.traceControl -i process identifier -enable
   ```
   Trace is now turned on for the application.

   When trace is enabled, the IBM MQ classes for JMS start writing trace information to a trace file in the application's current working directory. The name of the trace file depends on the environment that the application is running in:
   - For IBM MQ classes for JMS for Version 8.0.0, Fix Pack 6 or earlier, trace is written to a file called `mqjms_`*%PID%*`.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for JMS from the JAR file `com.ibm.mq.jar`, trace is written to a file called `mqjava_`*%PID%*`.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for JMS from the relocatable JAR file `com.ibm.mq.allclient.jar`, trace is written to a file called `mqjavaclient_`*%PID%*`.trc`.

   where *%PID%* is the process identifier of the application that is being traced.
4. To turn trace off, run the command:
   ```
   java -jar com.ibm.mq.traceControl -i process identifier -disable
   ```

Tracing using `MQJMS_TRACE_LEVEL`

To maintain backwards compatibility, the trace parameters used by Version 6.0 of IBM MQ classes for JMS are still supported. **`MQJMS_TRACE_LEVEL`** is deprecated for any new application.

This information has been removed. For information on how to enable trace, see "Tracing IBM MQ classes for JMS applications" on page 1709.

Tracing IBM MQ classes for Java applications

The trace facility in IBM MQ classes for Java is provided to help IBM Support to diagnose customer issues. Various properties control the behavior of this facility.

About this task

If you are asked to provide trace output to investigate an issue, use one of the options mentioned below:

- If the issue is easy to recreate, then collect an IBM MQ classes for Java trace by using a Java System Property. For more information, see "Collecting an IBM MQ classes for Java trace by using a Java system property" on page 1714.
- If an application needs to run for a period of time before the issue occurs, collect an IBM MQ classes for Java trace by using the IBM MQ classes for Java configuration file. For more information, see "Collecting an IBM MQ classes for Java trace by using the IBM MQ classes for Java configuration file" on page 1715.
- To generate a trace from an application that is currently running, collect the IBM MQ classes for Java trace dynamically by using the traceControl utility. For more information, see "Collecting an IBM MQ classes for Java trace dynamically by using the traceControl utility" on page 1716.

If you are unsure which option to use, contact your IBM Support representative and they will be able to advise you on the best way to collect trace for the issue you are seeing.

If a severe or unrecoverable error occurs, First Failure Support Technology (FFST) information is recorded in a file with a name of the format `JAVACC` *`xxxx`*`.FDC` where *`xxxx`* is a four-digit number. It is incremented to differentiate `.FDC` files.

`.FDC` files are always written to a subdirectory called `FFDC`. The subdirectory is in one of two locations, depending on whether trace is active:

Trace is active, and *`traceOutputName`* is set
: The FFDC directory is created as a subdirectory of the directory to which the trace file is being written.

Trace is not active or *`traceOutputName`* is not set
: The FFDC directory is created as a subdirectory of the current working directory.

The JSE common services uses `java.util.logging` as its trace and logging infrastructure. The root object of this infrastructure is the LogManager. The log manager has a reset method, which closes all handlers and sets the log level to `null`, which in effect turns off all the trace. If your application or application server calls java.util.logging.LogManager.getLogManager().reset(), it closes all trace, which might prevent you from diagnosing any problems. To avoid closing all trace, create a LogManager class with an overridden reset() method that does nothing, as in the following example.

```
package com.ibm.javaut.tests;
import java.util.logging.LogManager;
public class JmsLogManager extends LogManager {
        // final shutdown hook to ensure that the trace is finally shutdown
        // and that the lock file is cleaned-up
        public class ShutdownHook extends Thread{
                public void run(){
                        doReset();
                }
        }
                public JmsLogManager(){
                // add shutdown hook to ensure final cleanup
                Runtime.getRuntime().addShutdownHook(new ShutdownHook());
        }
                public void reset() throws SecurityException {
                // does nothing
        }
        public void doReset(){
                super.reset();
        }
        }
```

The shutdown hook is necessary to ensure that trace is properly shutdown when the JVM finishes. To use the modified log manager instead of the default one, add a system property to the JVM startup:

`java -Djava.util.logging.manager=com.` *`mycompany`*`.logging.LogManager ...`

Collecting an IBM MQ classes for Java trace by using a Java system property

For issues that can be reproduced in a short amount of time, IBM MQ classes for Java trace should be collected by setting a Java system property when starting the application.

About this task

To collect a trace by using a Java system property, complete the following steps.

Procedure

Run the application that is going to be traced by using the following command:

`java -Dcom.ibm.msg.client.commonservices.trace.status=ON` *`application_name`*

When the application starts, the IBM MQ classes for Java start writing trace information to a trace file in the application's current working directory. The name of the trace file depends on the environment that the application is running in:

- For IBM MQ classes for Java for Version 8.0.0, Fix Pack 6 or earlier, trace is written to a file called `mqjms_`*`%PID%`*`.trc`.
- V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for Java from the JAR file `com.ibm.mq.jar`, trace is written to a file called `mqjava_`*`%PID%`*`.trc`.
- V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for Java from the relocatable JAR file `com.ibm.mq.allclient.jar`, trace is written to a file called `mqjavaclient_`*`%PID%`*`.trc`.

where *%PID%* is the process identifier of the application that is being traced.
The application stops writing information to the trace file when it is stopped.
If the application has to run for a long period of time before the issue that the trace is being collected for occurs, then the trace file could potentially be very large. In this situation, consider collecting trace by using the IBM MQ classes for Java configuration file (see "Collecting an IBM MQ classes for Java trace by using the IBM MQ classes for Java configuration file" on page 1715). When enabling trace in this way, it is possible to control the amount of trace data that the IBM MQ classes for Java generates.

Collecting an IBM MQ classes for Java trace by using the IBM MQ classes for Java configuration file

If an application must run for a long period of time before an issue occurs, IBM MQ classes for Java trace should be collected by using the IBM MQ classes for Java configuration file. The configuration file allows you to specify various options to control the amount of trace data that is collected.

About this task

To collect a trace by using the IBM MQ classes for Java configuration file, complete the following steps.

Procedure

1. Create an IBM MQ classes for Java configuration file. For more information about this file, see The IBM MQ classes for Java configuration file.
2. Edit the IBM MQ classes for Java configuration file so that the property **`com.ibm.msg.client.commonservices.trace.status`** is set to the value `ON`.
3. Optional: Edit the other properties that are listed in the IBM MQ classes for Java configuration file Java Standard Edition Trace Settings.
4. Run the IBM MQ classes for Java application by using the following command:

   ```
   java -Dcom.ibm.msg.client.config.location=config_file_url
   application_name
   ```

   where *config_file_url* is a uniform resource locator (URL) that specifies the name and location of the IBM MQ classes for Java configuration file. URLs of the following types are supported: `http`, `file`, `ftp`, and `jar`.

   Here is an example of a Java command:

   ```
   java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myJava.config
   MyAppClass
   ```

   This command identifies the IBM MQ classes for Java configuration file as the file `D:\mydir\myJava.config` on the local Windows system.

   By default, the IBM MQ classes for Java start writing trace information to a trace file in the application's current working directory when the application starts up. The name of the trace file depends on the environment that the application is running in:

   - For IBM MQ classes for Java for Version 8.0.0, Fix Pack 6 or earlier, trace is written to a file called `mqjms_%PID%.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for Java from the JAR file `com.ibm.mq.jar`, trace is written to a file called `mqjava_%PID%.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for Java from the relocatable JAR file `com.ibm.mq.allclient.jar`, trace is written to a file called `mqjavaclient_%PID%.trc`.

   where *%PID%* is the process identifier of the application that is being traced.

   To change the name of the trace file, and the location where it is written, ensure that the IBM MQ classes for Java configuration file that the application uses contains an entry for the property **`com.ibm.msg.client.commonservices.trace.outputName`**. The value for the property can be either of the following:

   - The name of the trace file that is created in the application's working directory.
   - The fully qualified name of the trace file, including the directory in which the file is created.

   For example, to configure the IBM MQ classes for Java to write trace information for an application to a file called `C:\Trace\trace.trc`, the IBM MQ classes for Java configuration file that the application uses needs to contain the following entry:

   ```
   com.ibm.msg.client.commonservices.trace.outputName=C:\Trace\trace.trc
   ```

Collecting an IBM MQ classes for Java trace dynamically by using the traceControl utility

The traceControl utility that is shipped with the IBM MQ classes for Java allows trace to be collected from a running application. This can be very useful if IBM Support need to see a trace from an application once an issue has occurred, or if trace needs to be collected from a critical application that cannot be stopped.

About this task

For more information about the traceControl utility, see "Controlling trace in a running process by using IBM MQ classes for Java and IBM MQ classes for JMS" on page 1721.

To collect a trace by using the traceControl utility, complete the following steps.

Procedure

1. Bring up a command prompt, and navigate to the directory `MQ_INSTALLATION_PATH\java\lib`.
2. Run the command:
   ```
   java -jar com.ibm.mq.traceControl -list
   ```
   This command brings up a list of all of the Java processes on the system.
3. Identify the process identifier for the IBM MQ classes for Java application that needs to be traced, and run the command:
   ```
   java -jar com.ibm.mq.traceControl -i process identifier -enable
   ```
   Trace is now turned on for the application.

   When trace is enabled, the IBM MQ classes for Java start writing trace information to a trace file in the application's current working directory. The name of the trace file depends on the environment that the application is running in:
   - For IBM MQ classes for Java for Version 8.0.0, Fix Pack 6 or earlier, trace is written to a file called `mqjms_%PID%.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for Java from the JAR file `com.ibm.mq.jar`, trace is written to a file called `mqjava_%PID%.trc`.
   - V 8.0.0.7 From Version 8.0.0, Fix Pack 7, if the application has loaded the IBM MQ classes for Java from the relocatable JAR file `com.ibm.mq.allclient.jar`, trace is written to a file called `mqjavaclient_%PID%.trc`.

   where *%PID%* is the process identifier of the application that is being traced.
4. To turn trace off, run the command:
   ```
   java -jar com.ibm.mq.traceControl -i process identifier -disable
   ```

Tracing the IBM MQ Resource Adapter

The ResourceAdapter object encapsulates the global properties of the IBM MQ resource adapter. To enable trace of the IBM MQ resource adapter, properties need to be defined in the ResourceAdapter object.

The ResourceAdapter object has two sets of properties:

- Properties associated with diagnostic tracing
- Properties associated with the connection pool managed by the resource adapter

The way you define these properties depends on the administration interfaces provided by your application server.

Table 162 lists the properties of the ResourceAdapter object that are associated with diagnostic tracing.

Table 162. Properties of the ResourceAdapter object that are associated with diagnostic tracing

Name of property	**Type**	**Default value**	**Description**
traceEnabled	String	false	A flag to enable or disable diagnostic tracing. If the value is false, tracing is turned off.
traceLevel	String	3	The level of detail in a diagnostic trace. The value can be in the range 0, which produces no trace, to 10, which provides the most detail. See Table 163 for a description of each level.
logWriterEnabled	String	true	A flag to enable or disable the sending of a diagnostic trace to a LogWriter object provided by the application server. If the value is true, the trace is sent to a LogWriter object. If the value is false, any LogWriter object provided by the application server is not used.

Table 163 describes the levels of detail for diagnostic tracing.

Table 163. The levels of detail for diagnostic tracing

Level number	**Level of detail**
0	No trace.
1	The trace contains error messages.
3	The trace contains error and warning messages.
6	The trace contains error, warning, and information messages.
8	The trace contains error, warning, and information messages, and entry and exit information for methods.
9	The trace contains error, warning, and information messages, entry and exit information for methods, and diagnostic data.
10	The trace contains all trace information.

Note: Any level that is not included in this table is equivalent to the next lowest level. For example, specifying a trace level of 4 is equivalent to specifying a trace level of 3. However, the levels that are not included might be used in future releases of the IBM MQ resource adapter, so it is better to avoid using these levels.

If diagnostic tracing is turned off, error and warning messages are written to the system error stream. If diagnostic tracing is turned on, error messages are written to the system error stream and to the trace destination, but warning messages are written only to the trace destination. However, the trace contains warning messages only if the trace level is 3 or higher. By default, the trace destination is the current working directory, but if the logWriterEnabled property is set, the trace is sent to the application server.

In general, the ResourceAdapter object requires no administration. However, to enable diagnostic tracing on UNIX and Linux systems for example, you can set the following properties:

```
traceEnabled:    true
traceLevel:      10
```

These properties have no effect if the resource adapter has not been started, which is the case, for example, when applications using IBM MQ resources are running only in the client container. In this situation, you can set the properties for diagnostic tracing as Java Virtual Machine (JVM) system properties. You can set the properties by using the -D flag on the **java** command, as in the following example:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

You do not need to define all the properties of the ResourceAdapter object. Any properties left unspecified take their default values. In a managed environment, it is better not to mix the two ways of specifying properties. If you do mix them, the JVM system properties take precedence over the properties of the ResourceAdapter object.

Tracing additional IBM MQ Java components

For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Diagnostic information in this context consists of trace, first-failure data capture (FFDC) and error messages.

You can choose to have this information produced using IBM MQ facilities or the facilities of IBM MQ classes for Java or IBM MQ classes for JMS, as appropriate. Generally use the IBM MQ diagnostic facilities if they are available on the local system.

You might want to use the Java diagnostics in the following circumstances:

- On a system on which queue managers are available, if the queue manager is managed separately from the software you are running.
- To reduce performance effect of IBM MQ trace.

To request and configure diagnostic output, two system properties are used when starting an IBM MQ Java process:

- System property com.ibm.mq.commonservices specifies a standard Java property file, which contains a number of lines which are used to configure the diagnostic outputs. Each line of code in the file is free-format, and is terminated by a new line character.
- System property com.ibm.mq.commonservices.diagid associates trace and FFDC files with the process which created them.

For information about using the com.ibm.mq.commonservices properties file to configure diagnostics information, see “Using com.ibm.mq.commonservices” on page 1719.

For instructions on locating trace information and FFDC files, see “Java trace and FFDC files” on page 1720.

Related concepts:

"Using trace on UNIX and Linux systems" on page 1688
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

IBM i "Using trace on IBM MQ server on IBM i" on page 1692
Use the TRCMQM command to start and stop tracing and specify the type of trace that you require.

z/OS "Using trace for problem determination on z/OS" on page 1697
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

"Tracing TLS and SSL: **runmqakm** and iKeyman and iKeycmd functions" on page 1708
How to request **runmqakm** tracing and iKeyman and iKeycmd tracing.

Related reference:

"Using trace on Windows" on page 1686
Use the **strmqtrc** and **endmqtrc** commands or the MQ Explorer interface to start and end tracing.

Using com.ibm.mq.commonservices

The com.ibm.mq.commonservices properties file contains the following entries relating to the output of diagnostics from the Java components of IBM MQ.

Note that case is significant in all these entries:

Diagnostics.Java= *options*
: Which components are traced using Java trace. Options are one or more of *explorer*, *soap*, and *wmqjavaclasses*, separated by commas, where "explorer" refers to the diagnostics from the IBM MQ Explorer, "soap" refers to the diagnostics from the running process within IBM MQ Transport for SOAP, and "wmqjavaclasses" refers to the diagnostics from the underlying IBM MQ Java classes. By default no components are traced.

Diagnostics.Java.Trace.Detail= *high* | *medium* | *low*
: Detail level for Java trace. The *high* and *medium* detail levels match those used in IBM MQ tracing but *low* is unique to Java trace. This property is ignored if Diagnostics.Java is not set. The default is *medium*.

Diagnostics.Java.Trace.Destination.File= *enabled* | *disabled*
: Whether Java trace is written to a file. This property is ignored if Diagnostics.Java is not set. The default is *disabled*.

Diagnostics.Java.Trace.Destination.Console= *enabled* | *disabled*
: Whether Java trace is written to the system console. This property is ignored if Diagnostics.Java is not set. The default is *disabled*.

Diagnostics.Java.Trace.Destination.Pathname= *dirname*
: The directory to which Java trace is written. This property is ignored if Diagnostics.Java is not set or Diagnostics.Java.Trace.Destination.File=disabled. On UNIX and Linux systems, the default is /var/mqm/trace if it is present, otherwise the Java console (System.err). On Windows, the default is the system console.

Diagnostics.Java.FFDC.Destination.Pathname= *dirname*
: The directory to which Java FFDC output is written. The default is the current working directory.

Diagnostics.Java.Errors.Destination.Filename= *filename*
: The fully qualified file name to which Java error messages are written. The default is AMQJAVA.LOG in the current working directory.

An example of a com.ibm.mq.commonservices properties file is given in Figure 126 on page 1720. Lines beginning with the number sign (#) are treated as comments.

```
#
# Java diagnostics for WebSphere MQ Transport for SOAP
# and the WebSphere MQ Java Classes are both enabled
#
Diagnostics.Java=soap,wmqjavaclasses
#
# High detail Java trace
#
Diagnostics.Java.Trace.Detail=high
#
# Java trace is written to a file and not to the console.
#
Diagnostics.Java.Trace.Destination.File=enabled
Diagnostics.Java.Trace.Destination.Console=disabled
#
# Directory for Java trace file
#
Diagnostics.Java.Trace.Destination.Pathname=c:\\tracedir
#
# Directory for First Failure Data Capture
#
Diagnostics.Java.FFDC.Destination.Pathname=c:\\ffdcdir
#
# Directory for error logging
#
Diagnostics.Java.Errors.Destination.Filename=c:\\errorsdir\\SOAPERRORS.LOG
#
```

Figure 126. Sample com.ibm.mq.commonservices properties file

A sample properties file, WMQSoap_RAS.properties, is also supplied as part of the " Java messaging and SOAP transport" install option.

Java trace and FFDC files

File name conventions for Java trace and FFDC files.

When Java trace is generated for IBM MQ Transport for SOAP, it is written to a file with a name of the format `AMQ.` *`diagid`*`.` *`counter`*`.TRC`. Here, *diagid* is the value of the system property com.ibm.mq.commonservices.diagid associated with this Java process, as described earlier in this section, and *counter* is an integer greater than or equal to 0. All letters in the name are in uppercase, matching the naming convention used for normal IBM MQ trace.

If com.ibm.mq.commonservices.diagid is not specified, the value of *diagid* is the current time, in the format `YYYYMMDDhhmmssmmm`.

When Java trace is generated for the MQ Explorer, it is written to file with a name of the format `AMQYYYYMMDDHHmmssmmm.TRC.n`. Each time MQ Explorer trace is run, the trace facility renames all previous trace files by incrementing the file suffix `.n` by one. The trace facility then creates a new file with the suffix `.0` that is always the latest.

The IBM MQ Java classes trace file has a name based on the equivalent IBM MQ Transport for SOAP Java trace file. The name differs in that it has the string .JC added before the .TRC string, giving a format of `AMQ.` *`diagid`*`.` *`counter`*`.JC.TRC`.

When Java FFDC is generated for the MQ Explorer or for IBM MQ Transport for SOAP, it is written to a file with a name of the format `AMQ.` *`diagid`*`.` *`counter`*`.FDC` where *diagid* and *counter* are as described for Java trace files.

Java error message output for the MQ Explorer and for IBM MQ Transport for SOAP is written to the file specified by *Diagnostics.Java.Errors.Destination.Filename* for the appropriate Java process. The format of these files matches closely the format of the standard IBM MQ error logs.

When a process is writing trace information to a file, it appends to a single trace output file for the lifetime of the process. Similarly, a single FFDC output file is used for the lifetime of a process.

All trace output is in the UTF-8 character set.

Controlling trace in a running process by using IBM MQ classes for Java and IBM MQ classes for JMS

The IBM MQ classes for Java and IBM MQ classes for JMS register a Standard MBean that allows suitable Java Management Extensions (JMX) tools to control certain aspects of trace behavior for a client process.

Principles

As an alternative to the well-known general-purpose tools like `jconsole` you can use a command-line tool in the form of an executable JAR file to access these facilities.

The JAR file is called `com.ibm.mq.traceControl.jar` and is stored in the `java/lib` subdirectory of the IBM MQ installation.

Note: Depending on configuration, JMX tools can be used either locally (on the same system as the process) or remotely. The local case is discussed initially.

Finding the process

To control a process, you must establish a JMX connection it. To control a process locally, you must specify its identifier.

To display a summary of running Java processes with their identifiers, run the executable JAR file with the option `-list`. This option produces a list of identifiers and descriptions for the processes that are found.

Examining trace status

When you have found the identifier for the relevant process, run the executable JAR file with the options `-i identifier -status`, where 'identifier' is the identifier of the process you want to change. These options display the status, either `enabled` or `disabled`, for the process, and the information about where the process is running, the name of the trace file, and a tree that represents the inclusion and exclusion of packages in trace.

Enabling and disabling trace

To enable trace for a process, run the executable JAR file with the options `-i identifier -enable`.

To disable trace for a process, run the executable JAR file with the options `-i identifier -disable`.

Note: You can choose only one option from the set `-status`, `-enable`, and `-disable`.

Including and excluding packages

To include a package in trace for a process, run the executable JAR file with the options `-i identifier -ip` *`package_name`*, where *package_name* is the name of your package.

To exclude a package from trace for a process, run the executable JAR file with the options `-i identifier -ep` *package_name*.

Note: You can use multiple `-ip` and `-ep` options. These options are not checked for consistency.

When you specify a package for exclusion or inclusion, the handling of packages that have matching prefixes is not affected. For example, excluding the package com.ibm.mq.jms from trace would not exclude com.ibm.mq, com.ibm.msq.client.jms, or com.ibm.mq.remote.api, but it would exclude com.ibm.mq.jms.internal.

```
C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -list
10008 : 'MQSample'
9004 : ' MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -list'

C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -i 10008 -status
Tracing enabled : false
User Directory : C:\Users\IBM_ADMIN\RTCworkspace\sandpit
Trace File Name : mqjms.trc
Package Include/Exclude tree
root - Included

C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -i 10008 -enable
Enabling trace
Tracing enabled : true

C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -i 10008 -status
Tracing enabled : true
User Directory : C:\Users\IBM_ADMIN\RTCworkspace\sandpit
Trace File Name : mqjms_10008.trc
Package Include/Exclude tree
root - Included

C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -i 10008 -ip com.ibm.mq.jms
Adding 'com.ibm.mq.jms' to the list of packages included in trace


C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -i 10008 -status
Tracing enabled : true
User Directory : C:\Users\IBM_ADMIN\RTCworkspace\sandpit
Trace File Name : mqjms_10008.trc
Package Include/Exclude tree
root - Included
com - Included
ibm - Included
mq - Included
jms - Included

C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -i 10008 -ip com.acme.banana -ep com.acme.banana.spl
Adding 'com.acme.banana' to the list of packages included in trace
Adding 'com.acme.banana.shake' to the list of packages included in trace
Adding 'com.acme.banana.split' to the list of packages excluded from trace

C:>java -jar MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.traceControl.jar -i 10008 -status
Tracing enabled : true User Directory : C:\Users\IBM_ADMIN\RTCworkspace\sandpit
Trace File Name : mqjms_10008.trc
Package Include/Exclude tree
root - Included
com - Included
acme - Included
banana - Included
shake - Included
split - Excluded
ibm - Included
mq - Included
jms - Included
```

The package inclusion-exclusion tree

The tracing mechanism for IBM MQ classes for Java and IBM MQ classes for JMS tracks the inclusion and exclusion of packages by means of a tree structure, starting from a root node. In the tree structure each node represents one element of a package name, identified by the package name element and containing a trace status which can be either `Included` or `Excluded`. For example the package com.ibm.mq would be represented by three nodes identified by the strings com, ibm, and mq.

Initially, the tree usually contains entries to include most packages, but the header and pcf packages are excluded as they generate a lot of noise. So the initial tree will look something like

```
root - Included
com - Included
ibm - Included
mq - Included
headers - Excluded
pcf - Excluded
```

When the trace facility is determining whether to include or exclude a package, it matches leading portions of the package name to the nodes in the tree as far as possible and takes the status of the last matched node. At the initial state of the tree, the packages com.ibm.msg.client and com.ibm.mq.jms would be included, as the last nodes in the tree that matches them (com->ibm and com->ibm->mq respectively) are marked as *Included*. Conversely, the package com.ibm.headers.internal would be excluded as the last matching node in the tree (com->ibm->mq->headers) is marked as *Excluded*.

As further changes are made to the tree by using the `com.ibm.mq.TraceControl.jar`, it is important to remember that inclusion or exclusion only affects a package and child packages. So, given the initial state that is shown previously, specifying `-ep com.ibm.mq.jms`, would update the tree to look like:

```
root - Included
com - Included
ibm - Included
mq - Included
headers - Excluded
jms - Excluded
pcf - Excluded
```

Which would exclude packages com.ibm.mq.jms, and com.ibm.mq.jms.internal, without affecting packages outside the com.ibm.mq.jms.* hierarchy.

If `-ip com.ibm.mq.jms.admin` is specified next, the tree would look like:

```
root - Included
com - Included
ibm - Included
mq - Included
headers - Excluded
jms - Excluded
admin - Included
pcf - Excluded
```

Which would still exclude packages com.ibm.mq.jms, com.ibm.mq.jms.internal, but now the packages com.ibm.mq.jms.admin, and com.ibm.mq.jms.admin.internal are included in trace.

Connecting remotely

You can connect remotely only if the process was started with a JMX agent that is enabled for remote connection, and that uses the `-Dcom.sun.management.jmxremote.port=port_number` system setting.

After you have started with this system setting, you can run the executable JAR file with the options `-h` *`host_name`* `-p` *`port_number`* in place of the `-i` *`identifier`* option, where *host_name* is the name of the host you wish to connect to and *port_number* is the name of the port to be used.

Note: You must ensure that you take appropriate steps to minimize security risks by enabling SSL for the connection. See the Oracle documentation on JMX for further details http://www.oracle.com.

Limitations

The following limitations exist:

- For non-IBM JVMs, the tool must be started with `tools.jar` added to its class path. The command that is on these platforms is :

```
java -cp <MQ_INSTALL_DIR>/java/lib/com.ibm.mq.traceControl.jar;<JAVA_HOME>/lib/tools/jar com.ibm.msg.client.commonservice
```

- Local attach is controlled by user ID. The tool must be run under the same ID as the process that is to be controlled.

Problem determination on z/OS

z/OS

IBM MQ for z/OS, CICS, Db2, and IMS produce diagnostic information which can be used for problem determination.

This section contains information about the following topics:

- The recovery actions attempted by the queue manager when a problem is detected.
- IBM MQ for z/OS abends, and the information produced when an abend occurs.
- The diagnostic information produced by IBM MQ for z/OS, and additional sources of useful information.

The type of information provided to help with problem determination and application debugging depends on the type of error encountered, and the way your subsystem is set up.

See the following links for more information about problem determination and diagnostic information on IBM MQ for z/OS:

- "IBM MQ for z/OS performance constraints" on page 1725
- "IBM MQ for z/OS recovery actions" on page 1727
- "IBM MQ for z/OS abends" on page 1728
- "Diagnostic information produced on IBM MQ for z/OS" on page 1731
- "Other sources of information" on page 1733
- "Diagnostic aids for CICS" on page 1734
- "Diagnostic aids for IMS" on page 1735
- "Diagnostic aids for Db2" on page 1735
- "IBM MQ for z/OS dumps" on page 1735
- "Dealing with performance problems on z/OS" on page 1755
- "Dealing with incorrect output" on page 1762

Related concepts:

"Troubleshooting overview" on page 1276
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

"Using logs" on page 1678
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"First Failure Support Technology (FFST" on page 1666
First Failure Support Technology (FFST) for IBM MQ provides information about events that, in the case of an error, can help IBM support personnel to diagnose the problem.

Related tasks:

"Using trace" on page 1686
You can use different types of trace to help you with problem determination and troubleshooting.

IBM MQ for z/OS performance constraints

Use this topic to investigate z/OS resources that can cause performance constraints.

There are a number of decisions to be made when customizing IBM MQ for z/OS that can affect the way your systems perform. These decisions include:

- The size and placement of data sets
- The allocation of buffers
- The distribution of queues among page sets, and Coupling Facility structures
- The number of tasks that you allow to access the queue manager at any one time

Log buffer pools

Insufficient log buffers can cause applications to wait until a log buffer is available, which can affect IBM MQ performance. RMF reports might show heavy I/O to volumes that hold log data sets.

There are three parameters you can use to tune log buffers. The most important is OUTBUFF. If the log manager statistic QJSTWTB is greater than 0, increase the size of the log buffer. This parameter controls the number of buffers to be filled before they are written to the active log data sets (in the range 1 - 256). Commits and out-of-syncpoint processing of persistent messages cause log buffers to be written out to the log. As a result this parameter might have little effect except when processing large messages, and the number of commits or out of sync point messages is low. These parameters are specified in the CSQ6LOGP macro (see Using CSQ6LOGP for details), and the significant ones are:

OUTBUFF
This parameter controls the size of the output buffer (in the range 40 KB through 4000 KB).

WRTHRSH
This parameter controls the number of buffers to be filled before they are written to the active log data sets (in the range 1 through 256).

You must also be aware of the LOGLOAD parameter of the CSQ6SYSP macro. This parameter specifies the number of log records that are written between checkpoint records. The range is 200 through 16 000 000 but a typical value for a large system is 500 000. If a value is too small you receive frequent checkpoints, which consume processor time and can cause additional disk I/O.

Buffer pool size

There is a buffer pool associated with each page set. You can specify the number of buffers in the buffer pool using the DEFINE BUFFPOOL command.

Incorrect specification of buffer pool size can adversely affect IBM MQ performance. The smaller the buffer pool, the more frequently physical I/O is required. RMF might show heavy I/O to volumes that hold page sets. For buffer pools with only short-lived messages the buffer manager statistics QPSTSLA, QPSTSOS, and QPSTRIO must typically be zero. For other buffer pools, QPSTSOS and QPSTSTLA must be zero.

Distribution of data sets on available DASD

The distribution of page data sets on DASD can have a significant effect on the performance of IBM MQ.

Place log data sets on low usage volumes with log *n* and log *n+1* on different volumes. Ensure that dual logs are placed on DASD on different control units and that the volumes are not on the same physical disk.

Distribution of queues on page sets

The distribution of queues on page sets can affect performance. This change in performance can be indicated by poor response times experienced by transactions using specific queues that reside on heavily used page sets. RMF reports might show heavy I/O to volumes containing the affected page sets.

You can assign queues to specific page sets by defining storage class (STGCLASS) objects specifying a particular page set, and then defining the STGCLASS parameter in the queue definition. It is a good idea to define heavily used queues on different page sets in this way.

Distribution of queues on Coupling Facility structures

The distribution of queues on Coupling Facility structures can affect performance.

A queue-sharing group can connect to up to 64 Coupling Facility structures, one of which must be the administration structure. You can use the remaining 63 Coupling Facility structures for IBM MQ data with each structure holding up to 512 queues. If you need more than one Coupling Facility structure, separate the queues across several structures based on the function of the queue.

There are some steps you can take to maximize efficiency:
- Delete any Coupling Facility structures you no longer require.
- Place all the queues used by an application on the same Coupling Facility to make application processing efficient.
- If work is particularly performance sensitive, choose a faster Coupling Facility structure.

Consider that if you lose a Coupling Facility structure, you lose any non-persistent messages stored in it. The loss of these non-persistent messages can cause consistency problems if queues are spread across various Coupling Facility structures. To use persistent messages, you must define the Coupling Facility structures with at least CFLEVEL(3) and RECOVER(YES).

Limitation of concurrent threads

The number of tasks accessing the queue manager can also affect performance, particularly if there are other constraints, such as storage, or there are many tasks accessing a few queues. The symptoms can be heavy I/O against one or more page sets, or poor response times from tasks known to access the same queues. The number of threads in IBM MQ is limited to 32767 for both TSO and Batch.

In a CICS environment, you can use CICS MAXTASK to limit concurrent access.

Using the IBM MQ trace for administration

Although you might have to use specific traces on occasion, using the trace facility has a negative effect on the performance of your systems.

Consider what destination you want your trace information sent to. Using the internal trace table saves I/O, but it is not large enough for traces that produce large volumes of data.

The statistics trace gathers information at intervals. The intervals are controlled by the STATIME parameter of the CSQ6SYSP macro, described in Using CSQ6SYSP. An accounting trace record is produced when the task or channel ends, which might be after many days.

You can limit traces by class, resource manager identifier (RMID), and instrumentation facility identifier (IFCID) to reduce the volume of data collected. See START TRACE for more information.

IBM MQ for z/OS recovery actions

z/OS

Use this topic to understand some of the recovery actions for user detected and queue manager detected errors.

IBM MQ for z/OS can recover from program checks caused by incorrect user data. A completion and reason code are issued to the caller. These codes are documented in IBM MQ for z/OS messages, completion, and reason codes.

Program errors

Program errors might be associated with user application program code or IBM MQ code, and fall into two categories:
- User detected errors
- Subsystem detected errors

User detected errors

User detected errors are detected by the user (or a user-written application program) when the results of a service request are not as expected (for example, a nonzero completion code). The collection of problem determination data cannot be automated because detection occurs after the IBM MQ function has completed. Rerunning the application with the IBM MQ user parameter trace facility activated can provide the data needed to analyze the problem. The output from this trace is directed to the *generalized trace facility* (GTF).

You can turn the trace on and off using an operator command. See “Using trace for problem determination on z/OS” on page 1697 for more information.

Queue manager detected errors

The queue manager detects errors such as:
- A program check
- A data set filling up
- An internal consistency error

IBM MQ analyzes the error and takes the following actions:
- If the problem was caused by a user or application error (such as an invalid address being used), the error is reflected back to the application by completion and reason codes.

- If the problem was not caused by a user or application error (for example, all available DASD has been used, or the system detected an internal inconsistency), IBM MQ recovers if possible, either by sending completion and reason codes to the application, or if this is not possible, by stopping the application.
- If IBM MQ cannot recover, it terminates with a specific reason code. An SVC dump is typically taken recording information in the *system diagnostic work area* (SDWA) and *variable recording area* (VRA) portions of the dump, and an entry is made in SYS1.LOGREC.

IBM MQ for z/OS abends

Abends can occur in WebSphere for z/OS or other z/OS systems. Use this topic to understand the IBM MQ system abend codes and how to investigate abends which occur in CICS, IMS, and z/OS.

IBM MQ for z/OS uses two system abend completion codes, X'5C6' and X'6C6'. These codes identify:

- Internal errors encountered during operation
- Diagnostic information for problem determination
- Actions initiated by the component involved in the error

X'5C6'
An X'5C6' abend completion code indicates that IBM MQ has detected an internal error and has terminated an internal task (TCB) or a user-connected task abnormally. Errors associated with a X'5C6' abend completion code might be preceded by a z/OS system code, or by internal errors.

Examine the diagnostic material generated by the X'5C6' abend to determine the source of the error that actually resulted in a subsequent task or subsystem termination.

X'6C6'
An X'6C6' abend completion code indicates that IBM MQ has detected a severe error and has terminated the queue manager abnormally. When a X'6C6' is issued, IBM MQ has determined that continued operation could result in the loss of data integrity. Errors associated with a X'6C6' abend completion code might be preceded by a z/OS system error, one or more X'5C6' abend completion codes, or by error message CSQV086E indicating abnormal termination of IBM MQ.

Table 164 summarizes the actions and diagnostic information available to IBM MQ for z/OS when these abend completion codes are issued. Different pieces of this information are relevant in different error situations. The information produced for a particular error depends upon the specific problem. For more information about the z/OS services that provide diagnostic information, see “Diagnostic information produced on IBM MQ for z/OS” on page 1731.

Table 164. Abend completion codes

	X'5C6'	**X'6C6'**
Explanation	• Error during IBM MQ normal operation	• Severe error; continued operation might jeopardize data integrity
System action	• Internal IBM MQ task is abended • Connected user task is abended	• The entire IBM MQ subsystem is abended • User task with an active IBM MQ connection might be abnormally terminated with a X'6C6' code • Possible MEMTERM (memory termination) of connected allied address space
Diagnostic information	• SVC dump • SYS1.LOGREC entry • VRA data entries	• SYS1.LOGREC • VRA data entries

Table 164. Abend completion codes (continued)

	X'5C6'	**X'6C6'**
Associated reason codes	• IBM MQ abend reason code • Associated z/OS system codes	• Subsystem termination reason code • z/OS system completion codes and X'5C6' codes that precede the X'6C6' abend
Location of accompanying codes	• SVC dump title • Message CSQW050I • Register 15 of SDWA section 'General Purpose Registers at Time of Error' • SYS1.LOGREC entries • VRA data entries	• SYS1.LOGREC • VRA data entries • Message CSQV086E, which is sent to z/OS system operator

Related concepts:

"Dealing with program abends"
Abends can occur with applications and other z/OS systems. Use this topic to investigate the various types of abends that can occur when using IBM MQ for z/OS.

"CICS, IMS, and z/OS abends" on page 1730
Use this topic to investigate abends from CICS, IMS, and z/OS.

"Diagnostic information produced on IBM MQ for z/OS" on page 1731
Use this topic to investigate some of the diagnostic information produced by z/OS that can be useful in problem determination and understand how to investigate error messages, dumps, console logs, job output, symptom strings, and queue output.

"IBM MQ for z/OS dumps" on page 1735
Use this topic for information about the use of dumps in problem determination. It describes the steps you should take when looking at a dump produced by an IBM MQ for z/OS address space.

Dealing with program abends

Abends can occur with applications and other z/OS systems. Use this topic to investigate the various types of abends that can occur when using IBM MQ for z/OS.

Types of abend

Program abends can be caused by applications failing to check, and respond to, reason codes from IBM MQ. For example, if a message has not been received, using fields that would have been set up in the message for calculation might cause X'0C4' or X'0C7' abends (ASRA abends in CICS).

The following pieces of information indicate a program abend:

- Error messages from IBM MQ in the console log
- CICS error messages
- CICS transaction dumps
- IMS region dumps
- IMS messages on user or master terminal
- Program dump information in batch or TSO output
- Abend messages in batch job output
- Abend messages on the TSO screen

If you have an abend code, see one of the following manuals for an explanation of the cause of the abend:

- For IBM MQ for z/OS abends (abend codes X'5C6' and X'6C6'), see IBM MQ for z/OS messages, completion, and reason codes

- For batch abends, the *MVS System Codes* manual
- For CICS abends, the *CICS Messages and Codes* manual
- For IMS abends, the *IMS/ESA Messages and Codes* manual
- For Db2 abends, the *Db2 Messages and Codes* manual
- For RRS abends, the *MVS System Messages* manual
- For XES abends, the *MVS System Messages* manual

Batch abends

Batch abends cause an error message containing information about the contents of registers to be displayed in the syslog. TSO abends cause an error message containing similar information to be produced on the TSO screen. A SYSUDUMP is taken if there is a SYSUDUMP DD statement for the step (see "IBM MQ for z/OS dumps" on page 1735).

CICS transaction abends

CICS transaction abends are recorded in the CICS CSMT log, and a message is produced at the terminal (if there is one). A CICS AICA abend indicates a possible loop. See "Dealing with loops" on page 1760 for more information. If you have a CICS abend, using CEDF and the CICS trace might help you to find the cause of the problem. See the *CICS Problem Determination Guide* for more information.

IMS transaction abends

IMS transaction abends are recorded on the IMS master terminal, and an error message is produced at the terminal (if there is one). If you have an IMS abend, see the *IMS/ESA Diagnosis Guide and Reference* manual.

CICS, IMS, and z/OS abends

Use this topic to investigate abends from CICS, IMS, and z/OS.

CICS abends

A CICS abend message is sent to the terminal, if the application is attached to one, or to the CSMT log. CICS abend codes are explained in the *CICS Messages and Codes* manual.

The CICS adapter issues abend reason codes beginning with the letter Q (for example, QDCL). These codes are documented in IBM MQ for z/OS messages, completion, and reason codes

IMS abends

An IMS application might abend in one of the following circumstances:
- A normal abend.
- An IMS pseudo abend, with an abend code such as U3044 resulting from an error in an ESAF exit program.
- Abend 3051 or 3047, when the REO (region error option) has been specified as "Q" or "A", and an IMS application attempts to reference a non-operational external subsystem, or when resources are unavailable at the time when a thread is created.

An IMS message is sent to the user terminal or job output, and the IMS master terminal. The abend might be accompanied by a region dump.

z/OS abends

During IBM MQ operation, an abend might occur with a z/OS system completion code. If you receive a z/OS abend, see the appropriate z/OS publication.

Diagnostic information produced on IBM MQ for z/OS

z/OS

Use this topic to investigate some of the diagnostic information produced by z/OS that can be useful in problem determination and understand how to investigate error messages, dumps, console logs, job output, symptom strings, and queue output.

IBM MQ for z/OS functional recovery routines use z/OS services to provide diagnostic information to help you in problem determination.

The following z/OS services provide diagnostic information:

SVC dumps
The IBM MQ abend completion code X'5C6' uses the z/OS SDUMP service to create SVC dumps. The content and storage areas associated with these dumps vary, depending on the specific error and the state of the queue manager at the time the error occurred.

SYS1.LOGREC
Entries are requested in the SYS1.LOGREC data set at the time of the error using the z/OS SETRP service. The following are also recorded in SYS1.LOGREC:

- Subsystem abnormal terminations
- Secondary abends occurring in a recovery routine
- Requests from the recovery termination manager

Variable recording area (VRA) data
Data entries are added to the VRA of the SDWA by using a z/OS VRA defined key. VRA data includes a series of diagnostic data entries common to all IBM MQ for z/OS abend completion codes. Additional information is provided during initial error processing by the invoking component recovery routine, or by the recovery termination manager.

IBM MQ for z/OS provides unique messages that, together with the output of dumps, are aimed at providing sufficient data to allow diagnosis of the problem without having to try to reproduce it. This is known as first failure data capture.

Error messages

IBM MQ produces an error message when a problem is detected. IBM MQ diagnostic messages begin with the prefix CSQ. Each error message generated by IBM MQ is unique; that is, it is generated for one and only one error. Information about the error can be found in IBM MQ for z/OS messages, completion, and reason codes.

The first three characters of the names of IBM MQ modules are also usually CSQ. The exceptions to this are modules for C++ (IMQ), and the header files (CMQ). The fourth character uniquely identifies the component. These identifiers are listed in “IBM MQ component and resource manager identifiers” on page 1657. Characters five through eight are unique within the group identified by the first four characters.

Make sure that you have some documentation on application messages and codes for programs that were written at your installation, as well as viewing IBM MQ for z/OS messages, completion, and reason codes

There might be some instances when no message is produced, or, if one is produced, it cannot be communicated. In these circumstances, you might have to analyze a dump to isolate the error to a particular module. For more information about the use of dumps, see “IBM MQ for z/OS dumps” on page 1735.

Dumps

Dumps are an important source of detailed information about problems. Whether they are as the result of an abend or a user request, they allow you to see a snapshot of what was happening at the moment the dump was taken. “IBM MQ for z/OS dumps” on page 1735 contains guidance about using dumps to locate problems in your IBM MQ system. However, because they only provide a snapshot, you might need to use them with other sources of information that cover a longer period of time, such as logs.

Snap dumps are also produced for specific types of error in handling MQI calls. The dumps are written to the CSQSNAP DD.

Console logs and job output

You can copy console logs into a permanent data set, or print them as required. If you are only interested in specific events, you can select which parts of the console log to print.

Job output includes output produced from running the job, as well as that from the console. You can copy this output into permanent data sets, or print it as required. You might need to collect output for all associated jobs, for example CICS, IMS, and IBM MQ.

Symptom strings

Symptom strings display important diagnostic information in a structured format. When a symptom string is produced, it is available in one or more of the following places:
- On the z/OS system console
- In SYS1.LOGREC
- In any dump taken

Figure 127 shows an example of a symptom string.

```
PIDS/ 5655R3600 RIDS/CSQMAIN1 AB/S6C6 PRCS/0E30003
```

Figure 127. Sample symptom string

The symptom string provides a number of keywords that you can use to search the IBM software support database. If you have access to one of the optional search tools, you can search the database yourself. If you report a problem to the IBM support center, you are often asked to quote the symptom string. (For more information about searching the IBM software support database, See “Searching the IBM database for similar problems, and solutions” on page 1639.)

Although the symptom string is designed to provide keywords for searching the database, it can also give you a lot of information about what was happening at the time the error occurred, and it might suggest an obvious cause or a promising area to start your investigation. See “Building a keyword string” on page 1643 for more information about keywords.

Queue information

You can display information about the status of queues by using the operations and control panels. Alternatively you can enter the DISPLAY QUEUE and DISPLAY QSTATUS commands from the z/OS console.

Note: If the command was issued from the console, the response is copied to the console log, allowing the documentation to be kept together compactly.

Related concepts:

"Using trace for problem determination on z/OS" on page 1697
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

"Other sources of information"
Use this topic to investigate other sources of information for problem determination.

"Diagnostic aids for CICS" on page 1734
You can use the CICS diagnostic transactions to display information about queue manager tasks, and MQI calls. Use this topic to investigate these facilities.

"Diagnostic aids for IMS" on page 1735
Use this topic to investigate IMS diagnostic facilities.

"Diagnostic aids for Db2" on page 1735
Use this topic to investigate references for Db2 diagnostic tools.

Other sources of information

Use this topic to investigate other sources of information for problem determination.

You might find the following items of documentation useful when solving problems with IBM MQ for z/OS.

- Your own documentation
- Documentation for the products you are using
- Source listings and link-edit maps
- Change log
- System configuration charts
- Information from the DISPLAY CONN command

Your own documentation

Your own documentation is the collection of information produced by your organization about what your system and applications should do, and how they are supposed to do it. How much of this information you need depends on how familiar you are with the system or application in question, and could include:

- Program descriptions or functional specifications
- Flowcharts or other descriptions of the flow of activity in a system
- Change history of a program
- Change history of your installation
- Statistical and monitoring profile showing average inputs, outputs, and response times

Documentation for the products you are using

The documentation for the product you are using are the InfoCenters in the IBM MQ library, and in the libraries for any other products you use with your application.

Make sure that the level of any documentation you refer to matches the level of the system you are using. Problems often arise through using either obsolete information, or information about a level of a product that is not yet installed.

Source listings and link-edit maps

Include the source listings of any applications written at your installation with your set of documentation. (They can often be the largest single element of documentation.) Make sure that you include the relevant output from the linkage editor with your source listings to avoid wasting time trying to find your way through a load module with an out-of-date link map. Be sure to include the JCL at the beginning of your listings, to show the libraries that were used and the load library the load module was placed in.

Change log

The information in the change log can tell you of changes made in the data processing environment that might have caused problems with your application program. To get the most out of your change log, include the data concerning hardware changes, system software (such as z/OS and IBM MQ) changes, application changes, and any modifications made to operating procedures.

System configuration charts

System configuration charts show what systems are running, where they are running, and how the systems are connected to each other. They also show which IBM MQ, CICS, or IMS systems are test systems and which are production systems.

Information from the DISPLAY CONN command

The DISPLAY CONN command provides information about which applications are connected to a queue manager, and information to help you to diagnose those that have a long-running unit of work. You could collect this information periodically and check it for any long-running units of work, and display the detailed information about that connection.

Diagnostic aids for CICS

You can use the CICS diagnostic transactions to display information about queue manager tasks, and MQI calls. Use this topic to investigate these facilities.

You can use the CKQC transaction (the CICS adapter control panels) to display information about queue manager tasks, and what state they are in (for example, a GET WAIT). See Administering IBM MQ for z/OS for more information about CKQC.

The application development environment is the same as for any other CICS application, and so you can use any tools normally used in that environment to develop IBM MQ applications. In particular, the *CICS execution diagnostic facility* (CEDF) traps entry to and exit from the CICS adapter for each MQI call, as well as trapping calls to all CICS API services. Examples of the output produced by this facility are given in Examples of CEDF output.

The CICS adapter also writes trace entries to the CICS trace. These entries are described in "CICS adapter trace entries" on page 1705.

Additional trace and dump data is available from the CICS region. These entries are as described in the *CICS Problem Determination Guide*.

Diagnostic aids for IMS

Use this topic to investigate IMS diagnostic facilities.

The application development environment is the same as for any other IMS application, and so any tools normally used in that environment can be used to develop IBM MQ applications.

Trace and dump data is available from the IMS region. These entries are as described in the *IMS/ESA Diagnosis Guide and Reference* manual.

Diagnostic aids for Db2

Use this topic to investigate references for Db2 diagnostic tools.

Refer to the following manuals for help in diagnosing Db2 problems:
- *Db2 for z/OS Diagnosis Guide and Reference*
- *Db2 Messages and Codes*

IBM MQ for z/OS dumps

z/OS

Use this topic for information about the use of dumps in problem determination. It describes the steps you should take when looking at a dump produced by an IBM MQ for z/OS address space.

How to use dumps for problem determination

When solving problems with your IBM MQ for z/OS system, you can use dumps in two ways:
- To examine the way IBM MQ processes a request from an application program.

  To do this, you typically need to analyze the whole dump, including control blocks and the internal trace.
- To identify problems with IBM MQ for z/OS itself, under the direction of IBM support center personnel.

Use the instructions in the following topics to get and process a dump:
- “Getting a dump” on page 1736
- “Using the z/OS DUMP command” on page 1737
- “Processing a dump using the IBM MQ for z/OS dump display panels” on page 1738
- “Processing a dump using line mode IPCS” on page 1742
- “Processing a dump using IPCS in batch” on page 1750

The dump title might provide sufficient information in the abend and reason codes to resolve the problem. You can see the dump title in the console log, or by using the z/OS command `DISPLAY DUMP,TITLE`. The format of the dump title is explained in “Analyzing the dump and interpreting dump titles” on page 1750. For information about the IBM MQ for z/OS abend codes, see “IBM MQ for z/OS abends” on page 1728, and abend reason codes are documented in IBM MQ for z/OS messages, completion, and reason codes.

If there is not enough information about your problem in the dump title, format the dump to display the other information contained in it.

See the following topics for information about different types of dumps:
- “SYSUDUMP information” on page 1752
- “Snap dumps” on page 1753

- “SYS1.LOGREC information” on page 1754
- “SVC dumps” on page 1754

Related concepts:

“Using trace for problem determination on z/OS” on page 1697
There are different trace options that can be used for problem determination with IBM MQ. Use this topic to understand the different options and how to control trace.

“IBM MQ for z/OS abends” on page 1728
Abends can occur in WebSphere for z/OS or other z/OS systems. Use this topic to understand the IBM MQ system abend codes and how to investigate abends which occur in CICS, IMS, and z/OS.

“Diagnostic information produced on IBM MQ for z/OS” on page 1731
Use this topic to investigate some of the diagnostic information produced by z/OS that can be useful in problem determination and understand how to investigate error messages, dumps, console logs, job output, symptom strings, and queue output.

Getting a dump

Use this topic to understand the different dump types for problem determination.

The following table shows information about the types of dump used with IBM MQ for z/OS and how they are initiated. It also shows how the dump is formatted:

Table 165. Types of dump used with IBM MQ for z/OS

Dump type	Data set	Output type	Formatted by	Caused by
SVC	Defined by system	Machine readable	IPCS in conjunction with an IBM MQ for z/OS verb exit	z/OS or IBM MQ for z/OS functional recovery routine detecting error, or the operator entering the z/OS DUMP command
SYSUDUMP	Defined by JCL (SYSOUT=A)	Formatted	Normally SYSOUT=A	An abend condition (only taken if there is a SYSUDUMP DD statement for the step)
Snap	Defined by JCL CSQSNAP (SYSOUT=A)	Formatted	Normally SYSOUT=A	Unexpected MQI call errors reported to adapters, or FFST information from the channel initiator
Stand-alone	Defined by installation (tape or disk)	Machine readable	IPCS in conjunction with an IBM MQ for z/OS verb exit	Operator IPL of the stand-alone dump program

IBM MQ for z/OS recovery routines request SVC dumps for most X'5C6' abends. The exceptions are listed in “SVC dumps” on page 1754. SVC dumps issued by IBM MQ for z/OS are the primary source of diagnostic information for problems.

If the dump is initiated by the IBM MQ subsystem, information about the dump is put into area called the *summary portion*. This contains information that the dump formatting program can use to identify the key components.

For more information about SVC dumps, see the *MVS Diagnosis: Tools and Service Aids* manual.

Using the z/OS DUMP command

To resolve a problem, IBM can ask you to create a dump file of the queue manager address space, channel initiator address space, or coupling facilities structures. Use this topic to understand the commands to create these dump files.

You might be asked to create dump file for any or several of the following items for IBM to resolve the problem:

- Main IBM MQ address space
- Channel initiator address space
- Coupling facility application structure
- Coupling facility administration structure for your queue-sharing group

Figure 128 through to Figure 132 on page 1738 show examples of the z/OS commands to do this, assuming a subsystem name of CSQ1.

```
DUMP COMM=(MQ QUEUE MANAGER DUMP)
*01 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
R 01,JOBNAME=(CSQ1MSTR,BATCH),CONT
*02 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
 IEE600I REPLY TO 01 IS;JOBNAME=CSQ1MSTR,CONT
R 02,SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
 IEE600I REPLY TO 02 IS;SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
IEA794I SVC DUMP HAS CAPTURED: 869
DUMPID=001 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=MQ QUEUE MANAGER MAIN DUMP
```

Figure 128. Dumping the IBM MQ queue manager and application address spaces

```
DUMP COMM=(MQ QUEUE MANAGER DUMP)
*01 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
R 01,JOBNAME=(CSQ1MSTR),CONT
*02 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
 IEE600I REPLY TO 01 IS;JOBNAME=CSQ1MSTR,CONT
R 02,SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
 IEE600I REPLY TO 02 IS;SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
IEA794I SVC DUMP HAS CAPTURED: 869
DUMPID=001 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=MQ QUEUE MANAGER DUMP
```

Figure 129. Dumping the IBM MQ queue manager address space

```
DUMP COMM=(MQ CHIN DUMP)
*01 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
R 01,JOBNAME=CSQ1CHIN,CONT
*02 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
 IEE600I REPLY TO 01 IS;JOBNAME=CSQ1CHIN,CONT
R 02,SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),CONT
*03 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
 IEE600I REPLY TO 02 IS;SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),CONT
R 03,DSPNAME=('CSQ1CHIN'.CSQXTRDS),END
IEE600I REPLY TO 03 IS;DSPNAME='CSQ1CHIN'.CSQXTRDS,END
IEA794I SVC DUMP HAS CAPTURED: 869
DUMPID=001 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=MQ CHIN DUMP
```

Figure 130. Dumping the channel initiator address space

```
DUMP COMM=(MQ MSTR & CHIN DUMP)
*01 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
R 01,JOBNAME=(CSQ1MSTR,CSQ1CHIN),CONT
*02 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
 IEE600I REPLY TO 01 IS;JOBNAME=(CSQ1MSTR,CSQ1CHIN),CONT
R 02,SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),CONT
*03 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
 IEE600I REPLY TO 02 IS;SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),CONT
R 03,DSPNAME=('CSQ1CHIN'.CSQXTRDS),END
IEE600I REPLY TO 03 IS;DSPNAME=('CSQ1CHIN'.CSQXTRDS),END
IEA794I SVC DUMP HAS CAPTURED: 869
DUMPID=001 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=MQ MSTR & CHIN DUMP
```

Figure 131. Dumping the IBM MQ queue manager and channel initiator address spaces

```
DUMP COMM=('MQ APPLICATION STRUCTURE 1 DUMP')
01 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
R 01,STRLIST=(STRNAME=QSG1APPLICATION1,(LISTNUM=ALL,ADJUNCT=CAPTURE,ENTRYDATA=UNSER))
IEE600I REPLY TO 01 IS;STRLIST=(STRNAME=QSG1APPLICATION1,(LISTNUM=
IEA794I SVC DUMP HAS CAPTURED: 677
DUMPID=057 REQUESTED BY JOB (*MASTER*)
DUMP TITLE='MQ APPLICATION STRUCTURE 1 DUMP'
```

Figure 132. Dumping a coupling facility structure

Processing a dump using the IBM MQ for z/OS dump display panels

You can use commands available through IPCS panels to process dumps. Use this topic to understand the IPCS options.

IBM MQ for z/OS provides a set of panels to help you process dumps. The following section describes how to use these panels:

1. From the IPCS PRIMARY OPTION MENU, select **ANALYSIS - Analyze dump contents** (option 2).

   The IPCS MVS ANALYSIS OF DUMP CONTENTS panel is displayed.
2. Select **COMPONENT - MVS component data** (option 6).

   The IPCS MVS DUMP COMPONENT DATA ANALYSIS panel is displayed. The appearance of the panel depends on the products installed at your installation, but will be similar to the panel shown in IPCS MVS Dump Component Data Analysis panel:

```
---------------- IPCS MVS DUMP COMPONENT DATA ANALYSIS -------------
OPTION ===>                                               SCROLL ===

To display information, specify "S option name" or enter S to the
left of the option desired.  Enter ? to the left of an option to
display help regarding the component support.

  Name     Abstract
  ALCWAIT  Allocation wait summary
  AOMDATA  AOM analysis
  ASMCHECK Auxiliary storage paging activity
  ASMDATA  ASM control block analysis
  AVMDATA  AVM control block analysis
  COMCHECK Operator communications data
  CSQMAIN  WebSphere MQ dump formatter panel interface
  CSQWDMP  WebSphere MQ dump formatter
  CTRACE   Component trace summary
  DAEDATA  DAE header data
  DIVDATA  Data-in-virtual storage
```

Figure 133. IPCS MVS Dump Component Data Analysis panel

3. Select **CSQMAIN IBM MQ dump formatter panel interface** by typing s beside the line and pressing Enter.

   If this option is not available, it is because the member CSQ7IPCS is not present; you should see Configuring z/OS for more information about installing the IBM MQ for z/OS dump formatting member.

   Note: If you have already used the dump to do a preliminary analysis, and you want to reexamine it, select **CSQWDMP IBM MQ dump formatter** to display the formatted contents again, using the default options.
4. The IBM MQ for z/OS - DUMP ANALYSIS menu is displayed. Use this menu to specify the action that you want to perform on a system dump.

```
---------------IBM WebSphere MQ for z/OS - DUMP ANALYSIS----------------
 COMMAND ===>



      1 Display all dump titles 00 through 99
      2 Manage the dump inventory
      3 Select a dump

      4 Display address spaces active at time of dump
      5 Display the symptom string
      6 Display the symptom string and other related data
      7 Display LOGREC data from the buffer in the dump
      8 Format and display the dump

      9 Issue IPCS command or CLIST



(c) Copyright IBM Corporation 1993, 2005. All rights reserved.

  F1=Help    F3=Exit   F12=Cancel
```

5. Before you can select a particular dump for analysis, the dump you require must be present in the dump inventory. To ensure that this is so, perform the following steps:
   a. If you do not know the name of the data set containing the dump, specify option 1 - **Display all dump titles xx through xx**.

      This displays the dump titles of all the dumps contained in the SYS1.DUMP data sets (where xx is a number in the range 00 through 99). You can limit the selection of data sets for display by using the xx fields to specify a range of data set numbers.

If you want to see details of all available dump data sets, set these values to 00 and 99.

Use the information displayed to identify the dump you want to analyze.

b. If the dump has not been copied into another data set (that is, it is in one of the SYS1.DUMP data sets), specify option 2 - **Manage the dump inventory**

   The dump inventory contains the dump data sets that you have used. Because the SYS1.DUMP data sets are reused, the name of the dump that you identified in step 5a on page 1739 might be in the list displayed. However, this entry refers to the previous dump that was stored in this data set, so delete it by typing DD next to it and pressing Enter. Then press F3 to return to the DUMP ANALYSIS MENU.

6. Specify option 3 - **Select a dump**, to select the dump that you want to work with. Type the name of the data set containing the dump in the Source field, check that NOPRINT and TERMINAL are specified in the Message Routing field (this is to ensure that the output is directed to the terminal), and press Enter. Press F3 to return to the DUMP ANALYSIS MENU.
7. Having selected a dump to work with, you can now use the other options on the menu to analyze the data in different parts of the dump:
   - To display a list of all address spaces active at the time the dump was taken, select option 4.
   - To display the symptom string, select option 5. If you want to use the symptom string to search the RETAIN database for solutions to similar problems, refer to “Searching the IBM database for similar problems, and solutions” on page 1639.
   - To display the symptom string and other serviceability information, including the variable recording area of the system diagnostic work area (SDWA), select option 6.
   - To format and display the data contained in the in-storage LOGREC buffer, select option 7.

     It could be that the abend that caused the dump was not the original cause of the error, but was caused by an earlier problem. To determine which LOGREC record relates to the cause of the problem, go to the bottom of the data set, type `FIND ERRORID: PREV`, and press Enter. The header of the latest LOGREC record is displayed, for example:

```
JOBNAME: NONE-FRR
 ERRORID: SEQ=00081  CPU=0040  ASID=0033  TIME=14:42:47.1

SEARCH ARGUMENT ABSTRACT

   PIDS/5655R3600 RIDS/CSQRLLM1#L RIDS/CSQRRHSL AB/S05C6
   PRCS/00D10231 REGS/0C1F0 RIDS/CSQVEUS2#R

   SYMPTOM            DESCRIPTION
   -------            -----------
   PIDS/5655R3600     PROGRAM ID: 5655R3600
.
.
.
```

     Note the program identifier (if it is not 5655R3600, the problem was not caused by IBM MQ for z/OS and you could be looking at the wrong dump). Also note the value of the TIME field. Repeat the command to find the previous LOGREC record, and note the value of the TIME field again. If the two values are close to each other (say, within about one or two tenths of a second), they could both relate to the same problem.

     You can use the symptom string from the LOGREC record related to the error to search the RETAIN database for solutions to similar problems (see “Searching the IBM database for similar problems, and solutions” on page 1639).
   - To format and display the dump, select option 8. The FORMAT AND DISPLAY THE DUMP panel is displayed:

```
---------IBM MQ for z/OS - FORMAT AND DISPLAY DUMP--------
COMMAND ===>

1 Display the control blocks and trace
2 Display just the control blocks
3 Display just the trace


Options:

Use the summary dump? . . . . . . . . . . . . . . __ 1 Yes
2 No


Subsystem name (required if summary dump not used) ____


Address space identifier or ALL. . . . . . . . . . ALL_



F1=Help  F3=Exit  F12=Cancel
```

- Use this panel to format your selected system dump. You can choose to display control blocks, data produced by the internal trace, or both, which is the default.

  Note: You cannot do this for dumps from the channel initiator, or for dumps of coupling facility structures.
  - To display the whole of the dump, that is:
    - The dump title
    - The variable recording area (VRA) diagnostic information report
    - The save area trace report
    - The control block summary
    - The trace table

    select option 1.
  - To display the information listed for option 1, without the trace table, select option 2.
  - To display the information listed for option 1, without the control blocks, select option 3.

  You can also use the following options:
  - **Use the Summary Dump?**

    Use this field to specify whether you want IBM MQ to use the information contained in the summary portion when formatting the selected dump. The default setting is YES.

    Note: If a summary dump has been taken, it might include data from more than one address space.
  - **Subsystem name**

    Use this field to identify the subsystem with the dump data you want to display. This is only required if there is no summary data (for example, if the operator requested the dump), or if you have specified NO in the **Use the summary dump?** field.

    If you do not know the subsystem name, type `IPCS SELECT ALL` at the command prompt, and press Enter to display a list of all the jobs running at the time of the error. If one of the jobs has the word ERROR against it in the SELECTION CRITERIA column, make a note of the name of that job. The job name is of the form *xxxx* MSTR, where *xxxx* is the subsystem name.

```
IPCS OUTPUT STREAM ------------------------
COMMAND ===>
ASID JOBNAME ASCBADDR SELECTION CRITERIA
---- -------- -------- ------------------
0001 *MASTER* 00FD4D80 ALL
0002 PCAUTH  00F8AB80 ALL
0003 RASP   00F8C100 ALL
0004 TRACE  00F8BE00 ALL
0005 GRS   00F8BC00 ALL
0006 DUMPSRV 00F8DE00 ALL
0008 CONSOLE 00FA7E00 ALL
0009 ALLOCAS 00F8D780 ALL
000A SMF   00FA4A00 ALL
000B VLF   00FA4800 ALL
000C LLA   00FA4600 ALL
000D JESM   00F71E00 ALL
001F MQM1MSTR 00FA0680 ERROR ALL
```

If no job has the word ERROR against it in the SELECTION CRITERIA column, select option 0 - DEFAULTS on the main IPCS Options Menu panel to display the IPCS Default Values panel. Note the address space identifier (ASID) and press F3 to return to the previous panel. Use the ASID to determine the job name; the form is *xxxx* MSTR, where *xxxx* is the subsystem name.

The following command shows which ASIDs are in the dump data set:

```
LDMP DSN('SYS1.DUMPxx') SELECT(DUMPED) NOSUMMARY
```

This shows the storage ranges dumped for each address space.

Press F3 to return to the FORMAT AND DISPLAY THE DUMP panel, and type this name in the **Subsystem name** field.

– **Address space identifier**

  Use this field if the data in a dump comes from more than one address space. If you only want to look at data from a particular address space, specify the identifier (ASID) for that address space.

  The default value for this field is ALL, which displays information about all the address spaces relevant to the subsystem in the dump. Change this field by typing the 4-character ASID over the value displayed.

  Note: Because the dump contains storage areas common to all address spaces, the information displayed might not be relevant to your problem if you specify the address space identifier incorrectly. In this case, return to this panel, and enter the correct address space identifier.

Related concepts:

"Processing a dump using line mode IPCS"
Use the IPCS commands to format a dump.

"Processing a dump using IPCS in batch" on page 1750
Use this topic to understand how IBM MQ dumps can be formatted by IPCS commands in batch mode.

"Analyzing the dump and interpreting dump titles" on page 1750
Use this topic to understand how dump titles are formatted, and how to analyze a dump.

Processing a dump using line mode IPCS

Use the IPCS commands to format a dump.

To format the dump using line mode IPCS commands, select the dump required by issuing the command:

```
SETDEF DSN('SYS1.DUMP xx ')
```

(where `SYS1.DUMP` *xx* is the name of the data set containing the dump). You can then use IPCS subcommands to display data from the dump.

See the following topics for information on how to format different types of dumps using IPCS commands:

- "Formatting an IBM MQ for z/OS dump"
- "Formatting a dump from the channel initiator" on page 1749

Related concepts:

"Processing a dump using the IBM MQ for z/OS dump display panels" on page 1738
You can use commands available through IPCS panels to process dumps. Use this topic to understand the IPCS options.

"Processing a dump using IPCS in batch" on page 1750
Use this topic to understand how IBM MQ dumps can be formatted by IPCS commands in batch mode.

"Analyzing the dump and interpreting dump titles" on page 1750
Use this topic to understand how dump titles are formatted, and how to analyze a dump.

Formatting an IBM MQ for z/OS dump:

Use this topic to understand how to format a queue manager dump using line mode IPCS commands.

The IPCS VERBEXIT CSQWDMP invokes the IBM MQ for z/OS dump formatting program (CSQWDPRD), and enables you to format an SVC dump to display IBM MQ data. You can restrict the amount of data that is displayed by specifying parameters.

IBM Service Personnel might require dumps of your coupling facility administration structure and application structures for your queue-sharing group, with dumps of queue managers in the queue-sharing group, to aid problem diagnosis. For information on formatting a coupling facility list structure, and the STRDATA subcommand, see the *MVS IPCS Commands* book.

Note: This section describes the parameters required to extract the necessary data. Separate operands by commas, not blanks. A blank that follows any operand in the control statement terminates the operand list, and any subsequent operands are ignored. Table 166 explains each keyword that you can specify in the control statement for formatting dumps.

Table 166. Keywords for the IBM MQ for z/OS dump formatting control statement

Keyword	Description
SUBSYS= *aaaa*	Use this keyword if the summary dump portion is not available, or not to be used, to give the name of the subsystem to format information for. *aaaa* is a 1 through 4-character subsystem name.
ALL (default)	All control blocks and the trace table.
AA	Data is displayed for all IBM MQ for z/OS control blocks in all address spaces.
DIAG=Y	Print diagnostic information. Use only under guidance from IBM service personnel. DIAG=N (suppresses the formatting of diagnostic information) is the default.
EB= *nnnnnnnn*	Only the trace points associated with this EB thread are displayed (the format of this keyword is EB= *nnnnnnnn* where *nnnnnnnn* is the 8-digit address of an EB thread that is contained in the trace). You must use this in conjunction with the TT keyword.
LG	All control blocks.
PTF=Y, LOAD= *load module name*	A list of PTFs at the front of the report (from MEPL). PTF=N (suppresses the formatting of such a list) is the default. The optional load subparameter allows you to specify the name of a load module, up to a maximum of 8 characters, for which to format a PTF report.

Table 166. Keywords for the IBM MQ for z/OS dump formatting control statement (continued)

Keyword	Description
SA= *hhhh*	The control blocks for a specified address space. Use either of the following formats: • SA= *hh* or • SA= *hhhh* where *h* represents a hexadecimal digit.
SG	A subset of system-wide control blocks.
TT ,HANDLES=x ,LOCKS=x ,INSYNCS=x ,URINFO=ALL/LONG	Format trace table Indicate threads with greater than x handles Indicate threads with greater than x locks Indicate threads with greater than x insync operations Show UR info for ALL threads or for long-running threads

Table 167details the dump formatting keywords that you can use to format the data relating to individual resource managers.

You cannot use these keywords in conjunction with any of the keywords inTable 166 on page 1743.

Table 167. Resource manager dump formatting keywords

Keyword	What is formatted

BMC=1
BMC=2(*buffer pool number*)
BMC=3(xx/yyyyyy)
BMC=4(xx/yyyyyy)
	Buffer manager data. BMC=1 formats control blocks of all buffers.
	BMC=2 formats data relating to the buffer identified in the 2-digit *buffer pool number*.
	BMC=3 and BMC=4 display a page from a pageset, if the page is present in a buffer. (The difference between BMC=3 and BMC=4 is the route taken to the page.)

BUFL= *<nnnnnnnnnnn>*	Storage access buffer allocation `sz`.
CALLD=Y =W	Show arrow for call depth in `TT`. and indent trace entry.
CALLTIME=Y	Print call time on exit trace.
CB= *(<addr>/[<strmodel>])*	Format address as IBM MQ block.
CBF=1	CBF report level 1.
CCB=S	Show the Composite Capability Block (CCB) for system EBs in TT.
CFS=1	CFS report level 1.
CFS=2	CFS report level 2.

CHLAUTH=1/2
ONAM=<20 chars>
	CHLAUTH report level.
	The optional ONAM subparameter allows you to specify the object name, up to a maximum of 20 characters, to limit data printed to objects starting with characters in ONAM.

CLXQ=1	Cluster XMITQ report level 1.

Table 167. Resource manager dump formatting keywords (continued)

Keyword	What is formatted

CLXQ=2
 	
ONAM=<20 chars>
	Cluster XMITQ report level 2.
	
	The optional ONAM subparameter allows you to specify the object name, up to a maximum of 20 characters, to limit data printed to objects starting with characters in ONAM.

CMD=0/1/2	Command trace table display level.
D=1/2/3	Detail level for some reports.
Db2=1	Db2 report level 1.

DMC=1,
 	
ONAM=<48 chars>
	DMC report level 1.
	
	The optional ONAM subparameter allows you to specify the object name, up to a maximum of 48 characters, to limit data printed to objects starting with characters in ONAM.

DMC=2,
 	
ONAM=<48 chars>
	DMC report level 2.
	
	The optional ONAM subparameter allows you to limit the objects printed to those with names beginning with the characters specified in ONAM (up to a maximum of 48 characters).

DMC=3,
 	
ONAM=<48 chars>
	DMC report level 3.
	
	The optional ONAM subparameter allows you to limit the objects printed to those with names beginning with the characters specified in ONAM (up to a maximum of 48 characters).

GR=1	Group indoubt report level 1.
IMS=1	IMS report level 1

Table 168. Resource manager dump formatting keywords (J -P)

Keyword	What is formatted
JOBNAME= *xxxxxxxx*	Job name
LKM=1	LKM report level 1.

Table 168. Resource manager dump formatting keywords (J -P) (continued)

Keyword	**What is formatted**
LKM=2/3, ,NAME=<up to 48 chars> ,NAMEX= *xxxxxxxxxxxxxxxx* ,NAMESP=1/2/3/4/5/6/7/8 ,TYPE=DMCP/QUALNM/TOPIC/ STGCLASS ,QUAL=GET/PUT/CRE/DFXQ/ PGSYNC/CHGCNT/ DELETE/EXPIRE LKM=3 LKM=4 ,JOBNAME= *xxxxxxxx* ,ASID= *xxxx*	LKM report level 2/3. Name (character) Name (Hex) Namespace Lock type Lock qualification LKM report level 3 LKM report level 4
LMC=1	LMC report level 1.
MAXTR= *nnnnnnnnn*	Max trace entries to format
MHASID= *xxxx*	Message handle ASID for properties
MMC=1 OBJ=MQLO/MQSH/MQRO/ MQAO/MQMO/MCHL/ MNLS/MSTC/MPRC/ : ' MAUT ONAM	MMC report level 1 Object type The optional ONAM subparameter allows you to limit the objects printed to those with names beginning with the characters specified in ONAM (up to a maximum of 48 characters).

MMC=2
 	
ONAM=<48 chars>
	MMC report level 2
	
	The optional ONAM subparameter allows you to limit the objects printed to those with names beginning with the characters specified in ONAM (up to a maximum of 48 characters).

MSG=*nnnnnnnnnnnnnnnn*
 	
MASID=*xxxx*
 	
LEN=*xxxxxxxx*
 	
MSGD=S/D
	Format the message at pointer.
	
	MASID allows storage in other address spaces.
	
	LEN limits amount of storage to format.
	
	MSGD controls level of detail.

MSGD=S/D	Message details in DMC=3, BMC=3/4, PSID reports. The parameter controls level of details, S is summary and D is detailed.
MSGH = *nnnnnnnnnnnnnnnn*	Message handle
MT	Message properties trace
MQVCX	MQCHARVs in hexadecimal format

Table 168. Resource manager dump formatting keywords (J -P) (continued)

Keyword	What is formatted
PROPS= *nnnnnnnnnnnnnnnn*	Message properties pointer
PSID= *nnnnnnnn*	Pageset to format page
PSTRX	Properties strings in hex format

Table 169. Resource manager dump formatting keywords (R -Z)

Keyword	What is formatted
RPR= *nnnnnnnn*	Page or record to format
SHOWDEL	Show deleted records for DMC=3
SMC=1/2/3	Storage manager
TC= * A E 0	TT data char format, concatenated print all in suitable character set always print ASCII always print EBCDIC never print either
TFMT=H/M	Time format - human or STCK
THR= *nnnnnnnn*	Thread address
THR=*/2/3	Set thread report level
TOP=1	TOP report level 1
TOP=2	TOP report level 2
TOP= *nnnnnnnnnnnnnnnn* /TSTR=<48 chars> /TSTRX=<hex 1208 str>	Tnode 64bit address or Topic string (wildcard with % at start or end)' This will be converted EBCDIC to ASCII, but only invariant characters Hexadecimal of topic string in 1208 always wildcard character at start.
TOP=3	TOP report level 3
TOP=4	TOP report level 4

TSEG=M(RU)/Q(P64)
 	
I(NTERPOLATE)
 	
F(WD)
 	
D(EBUG)
	Search process for 64-bit trace
	
	Guess missing TSEG address or addresses
	
	Force forward sort
	
	Debug search process

TSEG=(M,Q,I,F,D)	Specify multiple TSEG options
W=0/1/2/3	TT width format
XA=1	XA report level 1
ZMH = *nnnnnnnnnnnnnnnn*	ZST message handle

If the dump is initiated by the operator, there is no information in the summary portion of the dump.Table 170 on page 1748shows additional keywords that you can use in the CSQWDMP control statement.

Table 170. Summary dump keywords for the IBM MQ for z/OS dump formatting control statement

Keyword	Description
SUBSYS= *aaaa*	Use this keyword if the summary dump portion is not available, or not to be used, to give the name of the subsystem to format information for. *aaaa* is a 1 through 4-character subsystem name.
SUMDUMP=NO	Use this keyword if the dump has a summary portion, but you do not want to use it. (You would usually only do this if so directed by your IBM support center.)

The following list shows some examples of how to use these keywords:

- For default formatting of all address spaces, using information from the summary portion of the dump, use:
  ```
  VERBX CSQWDMP
  ```
- To display the trace table from a dump of subsystem named MQMT, which was initiated by an operator (and so does not have a summary portion) use:
  ```
  VERBX CSQWDMP 'TT,SUBSYS=MQMT'
  ```
- To display all the control blocks and the trace table from a dump produced by a subsystem abend, for an address space with ASID (address space identifier) 1F, use:
  ```
  VERBX CSQWDMP 'TT,LG,SA=1F'
  ```
- To display the portion of the trace table from a dump associated with a particular EB thread, use:
  ```
  VERBX CSQWDMP 'TT,EB= nnnnnnnn '
  ```
- To display message manager 1 report for local non-shared queue objects with a name begins with 'ABC' use:
  ```
  VERBX CSQWDMP 'MMC=1,ONAM=ABC,Obj=MQLO'
  ```

Table 171shows some other commands that are used frequently for analyzing dumps. For more information about these subcommands, see the *MVS IPCS Commands* manual.

Table 171. IPCS subcommands used for dump analysis

Subcommand	Description
STATUS	To display data usually examined during the initial part of the problem determination process.
STRDATA LISTNUM(ALL) ENTRYPOS(ALL) DETAIL	To format coupling facility structure data.
VERBEXIT LOGDATA	To format the in-storage LOGREC buffer records present before the dump was taken. LOGDATA locates the LOGREC entries that are contained in the LOGREC recording buffer and invokes the EREP program to format and print the LOGREC entries. These entries are formatted in the style of the normal detail edit report.
VERBEXIT TRACE	To format the system trace entries for all address spaces.
VERBEXIT SYMPTOM	To format the symptom strings contained in the header record of a system dump such as stand-alone dump, SVC dump, or an abend dump requested with a SYSUDUMP DD statement.
VERBEXIT GRSTRACE	To format diagnostic data from the major control blocks for global resource serialization.
VERBEXIT SUMDUMP	To locate and display the summary dump data that an SVC dump provides.
VERBEXIT DAEDATA	To format the dump analysis and elimination (DAE) data for the dumped system.

Related concepts:

"Formatting a dump from the channel initiator"
Use this topic to understand how to format a channel initiator dump using line mode IPCS commands.

Formatting a dump from the channel initiator:

Use this topic to understand how to format a channel initiator dump using line mode IPCS commands.

The IPCS VERBEXIT CSQXDPRD enables you to format a channel initiator dump. You can select the data that is formatted by specifying keywords.

This section describes the keywords that you can specify.

Table 172 describes the keywords that you can specify with CSQXDPRD.

Table 172. Keywords for the IPCS VERBEXIT CSQXDPRD

Keyword	What is formatted
SUBSYS= *aaaa*	The control blocks of the channel initiator associated with the named subsystem. It is required for all new formatted dumps.
CHST=1, CNAM= *channel name*, DUMP=S|F|C	All channel information. The optional CNAM subparameter allows you to specify the name of a channel, up to a maximum of 20 characters, for which to format details. The optional DUMP subparameter allows you to control the extent of formatting, as follows: • Specify DUMP=S (for "short") to format the first line of the hexadecimal dump of the channel data. • Specify DUMP=F (for "full") to format all lines of the data. • Specify DUMP=C (for "compressed⌂) to suppress the formatting of all duplicate lines in the data containing only X'00'. This is the default option
CHST=2, CNAM= *channel name*,	A summary of all channels, or of the channel specified by the CNAM keyword. See CHST=1 for details of the CNAM subparameter.
CHST=3, CNAM= *channel name*,	Data provided by CHST=2 and a program trace, line trace and formatted semaphore table print of all channels in the dump. See CHST=1 for details of the CNAM subparameter.
CLUS=1	Cluster report including the cluster repository known on the queue manager.
CLUS=2	Cluster report showing cluster registrations.

CTRACE=S|F,
 	
DPRO= *nnnnnnnn*,
 	
TCB= *nnnnnnn*
	Select either a short (CTRACE=S) or full (CTRACE=F) CTRACE.
	
	The optional DPRO subparameter allows you to specify a CTRACE for the DPRO specified.
	
	The optional TCB subparameter allows you to specify a CTRACE for the job specified.

DISP=1, DUMP=S|F|C	Dispatcher report See CHST=1 for details of the DUMP subparameter.
BUF=1	Buffer report
XSMF=1	Format channel initiator SMF data that is available in a dump.

Related concepts:

"Formatting an IBM MQ for z/OS dump" on page 1743
Use this topic to understand how to format a queue manager dump using line mode IPCS commands.

Processing a dump using IPCS in batch

Use this topic to understand how IBM MQ dumps can be formatted by IPCS commands in batch mode.

To use IPCS in batch, insert the required IPCS statements into your batch job stream (see Figure 134).

Change the data set name (DSN=) on the DUMP00 statement to reflect the dump you want to process, and insert the IPCS subcommands that you want to use.

```
//***********************************************
//*   RUNNING IPCS IN A BATCH JOB               *
//***********************************************
//MQMDMP   EXEC  PGM=IKJEFT01,REGION=5120K
//STEPLIB  DD    DSN=mqm.library-name,DISP=SHR
//SYSTSPRT DD    SYSOUT=*
//IPCSPRNT DD    SYSOUT=*
//IPCSDDIR DD    DSN=dump.directory-name,DISP=OLD
//DUMP00   DD    DSN=dump.name,DISP=SHR
//SYSTSIN  DD    *
  IPCS NOPARM TASKLIB(SCSQLOAD)
  SETDEF PRINT TERMINAL DDNAME(DUMP00) NOCONFIRM
  ************************************************
  * INSERT YOUR IPCS COMMANDS HERE, FOR EXAMPLE:   *
  VERBEXIT LOGDATA
  VERBEXIT SYMPTOM
  VERBEXIT CSQWDMP 'TT,SUBSYS=QMGR'
  ************************************************

  CLOSE   ALL
END
/*
```

Figure 134. Sample JCL for printing dumps through IPCS in the z/OS environment

Related concepts:

"Processing a dump using the IBM MQ for z/OS dump display panels" on page 1738
You can use commands available through IPCS panels to process dumps. Use this topic to understand the IPCS options.

"Processing a dump using line mode IPCS" on page 1742
Use the IPCS commands to format a dump.

"Analyzing the dump and interpreting dump titles"
Use this topic to understand how dump titles are formatted, and how to analyze a dump.

Analyzing the dump and interpreting dump titles

Use this topic to understand how dump titles are formatted, and how to analyze a dump.

- Analyzing the dump
- Dump title variation with PSW and ASID

Analyzing the dump

The dump title includes the abend completion and reason codes, the failing load module and CSECT names, and the release identifier. For more information on the dump title see Dump title variation with PSW and ASID

The formats of SVC dump titles vary slightly, depending on the type of error.

Figure 135 shows an example of an SVC dump title. Each field in the title is described after the figure.

```
ssnm,ABN=5C6-00D303F2,U=AUSER,C=R3600. 710.LOCK-CSQL1GET,
 M=CSQGFRCV,LOC=CSQLLPLM.CSQL1GET+0246
```

Figure 135. Sample SVC dump title

`ssnm,ABN=compltn-reason`

- `ssnm` is the name of the subsystem that issued the dump.
- `compltn` is the 3-character hexadecimal abend completion code (in this example, X'5C6'), prefixed by U for user abend codes.
- `reason` is the 4-byte hexadecimal reason code (in this example, X'00D303F2').

Note: The abend and reason codes might provide sufficient information to resolve the problem. See the IBM MQ for z/OS messages, completion, and reason codes for an explanation of the reason code.

`U=userid`

- `userid` is the user identifier of the user (in this example, AUSER). This field is not present for channel initiators.

`C=compid.release.comp-function`

- `compid` is the last 5 characters of the component identifier (explained in "The component-identifier keyword" on page 1645). The value R3600 uniquely identifies IBM MQ for z/OS.
- `release` is a 3-digit code indicating the version, release, and modification level of IBM MQ for z/OS (in this example, 710).
- `comp` is an acronym for the component in control at the time of the abend (in this example, LOCK).
- `function` is the name of a function, macro, or routine in control at the time of abend (in this example, CSQL1GET). This field is not always present.

`M=module`

- `module` is the name of the FRR or ESTAE recovery routine (in this example, CSQGFRCV). This field is not always present.

  Note: This is not the name of the module where the abend occurred; that is given by `LOC`.

`LOC=loadmod.csect+csect_offset`

- `loadmod` is the name of the load module in control at the time of the abend (in this example, CSQLLPLM). This might be represented by an asterisk if it is unknown.
- `csect` is the name of the CSECT in control at the time of abend (in this example, CSQL1GET).
- `csect_offset` is the offset within the failing CSECT at the time of abend (in this example, 0246).

Note: The value of `csect_offset` might vary if service has been applied to this CSECT, so do not use this value when building a keyword string to search the IBM software support database.

Dump title variation with PSW and ASID

Some dump titles replace the load module name, CSECT name, and CSECT offset with the PSW (program status word) and ASID (address space identifier). Figure 136 on page 1752 illustrates this format.

```
ssnm,ABN=compltn-reason,U=userid,C=compid.release.comp-function,
 M=module,PSW=psw_contents,ASID=address_space_id
```

Figure 136. Dump title with PSW and ASID

psw_contents

- The PSW at the time of the error (for example, X'077C100000729F9C').

address_space_id

- The address space in control at the time of the abend (for example, X'0011'). This field is not present for a channel initiator.

Related concepts:

"Processing a dump using the IBM MQ for z/OS dump display panels" on page 1738
You can use commands available through IPCS panels to process dumps. Use this topic to understand the IPCS options.

"Processing a dump using line mode IPCS" on page 1742
Use the IPCS commands to format a dump.

"Processing a dump using IPCS in batch" on page 1750
Use this topic to understand how IBM MQ dumps can be formatted by IPCS commands in batch mode.

SYSUDUMP information

The z/OS system can create SYSUDUMPs, which can be used as part of problem determination. This topic shows a sample SYSUDUMP output and gives a reference to the tools for interpreting SYSUDUMPs.

SYSUDUMP dumps provide information useful for debugging batch and TSO application programs. For more information about SYSUDUMP dumps, see the *MVS Diagnosis: Tools and Service Aids* manual.

Figure 137 on page 1753 shows a sample of the beginning of a SYSUDUMP dump.

```
JOB MQMBXBA1  STEP TSOUSER  TIME 102912   DATE 001019   ID = 000  CPUID = 632202333081   PAGE 00000001

COMPLETION CODE      SYSTEM = 0C1      REASON CODE = 00000001

PSW AT ENTRY TO ABEND  078D1000 000433FC        ILC 2  INTC 000D

PSW LOAD MODULE = BXBAAB01  ADDRESS = 000433FC  OFFSET = 0000A7F4

ASCB: 00F56400
+0000 ASCB..... ASCB      FWDP..... 00F60180  BWDP..... 0047800  CMSF..... 019D5A30  SVRB..... 008FE9E0
+0014 SYNC..... 00000D6F  IOSP..... 00000000  TNEW..... 00D18F0  CPUS..... 00000001  ASID..... 0066
+0026 R026..... 0000      LL5...... 00        HLHI..... 01       DPHI..... 00        DP....... 9D
+002C TRQP..... 80F5D381  LDA...... 7FF154E8  RSMF..... 00       R035..... 0000      TRQI..... 42
+0038 CSCB..... 00F4D048  TSB...... 00B61938  EJST..... 0000001  8C257E00

+0048 EWST..... 9CCDE747  76A09480            JSTL..... 00141A4  ECB...... 808FEF78  UBET..... 9CCDE740
  .
  .
  .
ASSB: 01946600
+0000 ASSB..... ASSB      VAFN..... 00000000  EVST..... 0000000  00000000

+0010 VFAT..... 00000000  00000000            RSV...... 000      XMCC..... 0000      XMCT.....00000000
+0020 VSC...... 00000000  NVSC..... 0000004C  ASRR..... 0000000  R02C..... 00000000  00000000 00000000
+0038           00000000  00000000

*** ADDRESS SPACE SWITCH EVENT MASK OFF (ASTESSEM = 0) ***

TCB: 008D18F0
+0000 RBP...... 008FE7D8  PIE...... 00000000  DEB...... 00B1530  TIO...... 008D4000  CMP......805C6000
+0014 TRN...... 40000000  MSS...... 7FFF7418  PKF...... 80       FLGS..... 01000000  00
+0022 LMP...... FF        DSP...... FE        LLS...... 00D1A88  JLB...... 00011F18  JPQ......00000000
+0030 GPRO-3... 00001000  008A4000  00000000  00000000
+0040 GPR4-7... 00FDC730  008A50C8  00000002  80E73F04
+0050 GPR8-11.. 81CC4360  008A6754  008A67B4  00000008
```

Figure 137. Sample beginning of a SYSUDUMP

Snap dumps

Snap dump data sets are controlled by z/OS JCL command statements. Use this topic to understand the CSQSNAP DD statement.

Snap dumps are always sent to the data set defined by the CSQSNAP DD statement. They can be issued by the adapters or the channel initiator.

- Snap dumps are issued by the batch, CICS, IMS, or RRS adapter when an unexpected error is returned by the queue manager for an MQI call. A full dump is produced containing information about the program that caused the problem.

  For a snap dump to be produced, the CSQSNAP DD statement must be in the batch application JCL, CICS JCL, or IMS dependent region JCL.
- Snap dumps are issued by the channel initiator in specific error conditions instead of a system dump. The dump contains information relating to the error. Message CSQX053E is also issued at the same time.

  To produce a snap dump, the CSQSNAP DD statement must be in the channel initiator started-task procedure.

SYS1.LOGREC information

Use this topic to understand how the z/OS SYS1.LOGREC information can assist with problem determination.

IBM MQ for z/OS and SYS1.LOGREC

The SYS1.LOGREC data set records various errors that different components of the operating system encounter. For more information about using SYS1.LOGREC records, see the *MVS Diagnosis: Tools and Service Aids* manual.

IBM MQ for z/OS recovery routines write information in the *system diagnostic work area* (SDWA) to the SYS1.LOGREC data set when retry is attempted, or when percolation to the next recovery routine occurs. Multiple SYS1.LOGREC entries can be recorded, because two or more retries or percolations might occur for a single error.

The SYS1.LOGREC entries recorded near the time of abend might provide valuable historical information about the events leading up to the abend.

Finding the applicable SYS1.LOGREC information

To obtain a SYS1.LOGREC listing, either:

- Use the EREP service aid, described in the *MVS Diagnosis: Tools and Service Aids* manual to format records in the SYS1.LOGREC data set.
- Specify the VERBEXIT LOGDATA keyword in IPCS.
- Use option 7 on the DUMP ANALYSIS MENU (refer to "Processing a dump using the IBM MQ for z/OS dump display panels" on page 1738).

Only records available in storage when the dump was requested are included. Each formatted record follows the heading `*****LOGDATA*****`.

SVC dumps

Use this topic to understand how to suppress SVC dumps, and reasons why SVC dumps are not produced.

When SVC dumps are not produced

Under some circumstances, SVC dumps are not produced. Generally, dumps are suppressed because of time or space problems, or security violations. The following list summarizes other reasons why SVC dumps might not be produced:

- The z/OS *serviceability level indication processing* (SLIP) commands suppressed the abend.

  The description of IEACMD00 in the *MVS Initialization and Tuning Reference* manual lists the defaults for SLIP commands executed at IPL time.
- The abend reason code was one that does not require a dump to determine the cause of abend.
- SDWACOMU or SDWAEAS (part of the system diagnostic work area, SDWA) was used to suppress the dump.

Suppressing IBM MQ for z/OS dumps using z/OS DAE

You can suppress SVC dumps that duplicate previous dumps. The *MVS Diagnosis: Tools and Service Aids* manual gives details about using z/OS *dump analysis and elimination* (DAE).

To support DAE, IBM MQ for z/OS defines two *variable recording area* (VRA) keys and a minimum symptom string. The two VRA keys are:

- KEY VRADAE (X'53'). No data is associated with this key.

- KEY VRAMINSC (X'52') DATA (X'08')

IBM MQ for z/OS provides the following data for the minimum symptom string in the *system diagnostic work area* (SDWA):

- Load module name
- CSECT name
- Abend code
- Recovery routine name
- Failing instruction area
- REG/PSW difference
- Reason code
- Component identifier
- Component subfunction

Dumps are considered duplicates for the purpose of suppressing duplicate dumps if eight (the X'08' from the VRAMINSC key) of the nine symptoms are the same.

Dealing with performance problems on z/OS

Use this topic to investigate performance problems in more detail.

Performance problems are characterized by the following:

- Poor response times in online transactions
- Batch jobs taking a long time to complete
- The transmission of messages is slow

Performance problems can be caused by many factors, from a lack of resource in the z/OS system as a whole, to poor application design.

The following topics present problems and suggested solutions, starting with problems that are relatively simple to diagnose, such as DASD contention, through problems with specific subsystems, such as IBM MQ and CICS or IMS.

- "IBM MQ for z/OS system considerations" on page 1756
- "CICS constraints" on page 1756
- "Dealing with applications that are running slowly or have stopped on z/OS" on page 1756

Remote queuing problems can be due to network congestion and other network problems. They can also be caused by problems at the remote queue manager.

Related concepts:

"Making initial checks" on page 1277
There are some initial checks that you can make that may provide answers to common problems that you may have.

"Dealing with incorrect output" on page 1762
Incorrect output can be missing, unexpected, or corrupted information. Read this topic to investigate further.

IBM MQ for z/OS system considerations

The z/OS system is an area that requires examination when investigating performance problems.

You might already be aware that your z/OS system is under stress because these problems affect many subsystems and applications.

You can use the standard monitoring tools such as Resource Monitoring Facility (RMF) to monitor and diagnose these problems. They might include:

- Constraints on storage (paging)
- Constraints on processor cycles
- Constraints on DASD
- Channel path usage

Use normal z/OS tuning techniques to resolve these problems.

CICS constraints

CICS constraints can also have an adverse effect on IBM MQ performance. Use this topic for further information about CICS constraints.

Performance of IBM MQ tasks can be affected by CICS constraints. For example, your system might have reached MAXTASK, forcing transactions to wait, or the CICS system might be short on storage. For example, CICS might not be scheduling transactions because the number of concurrent tasks has been reached, or CICS has detected a resource problem. If you suspect that CICS is causing your performance problems (for example because batch and TSO jobs run successfully, but your CICS tasks time out, or have poor response times), see the *CICS Problem Determination Guide* and the *CICS Performance Guide*.

Note: CICS I/O to transient data extrapartition data sets uses the z/OS RESERVE command. This could affect I/O to other data sets on the same volume.

Dealing with applications that are running slowly or have stopped on z/OS

Waits and loops can exhibit similar symptoms. Use the links in this topic to help differentiate between waits and loops.

Waits and loops are characterized by unresponsiveness. However, it can be difficult to distinguish between waits, loops, and poor performance.

Any of the following symptoms might be caused by a wait or a loop, or by a badly tuned or overloaded system:

- An application that appears to have stopped running (if IBM MQ for z/OS is still responsive, this problem is probably caused by an application problem)
- An MQSC command that does not produce a response
- Excessive use of processor time

To perform the tests shown in these topics, you need access to the z/OS console, and to be able to issue operator commands.

- "Distinguishing between waits and loops" on page 1757

- "Dealing with waits" on page 1758
- "Dealing with loops" on page 1760

Related concepts:

"Making initial checks" on page 1277
There are some initial checks that you can make that may provide answers to common problems that you may have.

Distinguishing between waits and loops:

Waits and loops on IBM MQ can present similar symptoms. Use this topic to help determine if you are experiencing a wait of a loop.

Because waits and loops can be difficult to distinguish, in some cases you need to carry out a detailed investigation before deciding which classification is right for your problem.

This section gives you guidance about choosing the best classification, and advice on what to do when you have decided on a classification.

Waits

For problem determination, a wait state is regarded as the state in which the execution of a task has been suspended. That is, the task has started to run, but has been suspended without completing, and has subsequently been unable to resume.

A problem identified as a wait in your system could be caused by any of the following:
- A wait on an MQI call
- A wait on a CICS or IMS call
- A wait for another resource (for example, file I/O)
- An ECB wait
- The CICS or IMS region waiting
- TSO waiting
- IBM MQ for z/OS waiting for work
- An apparent wait, caused by a loop
- Your task is not being dispatched by CICS or MVS due to higher priority work
- Db2 or RRS are inactive

Loops

A loop is the repeated execution of some code. If you have not planned the loop, or if you have designed it into your application but it does not terminate for some reason, you get a set of symptoms that vary depending on what the code is doing, and how any interfacing components and products react to it. In some cases, at first, a loop might be diagnosed as a wait or performance problem, because the looping task competes for system resources with other tasks that are not involved in the loop. However, a loop consumes resources but a wait does not.

An apparent loop problem in your system could be caused by any of the following:
- An application doing a lot more processing than usual and therefore taking much longer to complete
- A loop in application logic
- A loop with MQI calls
- A loop with CICS or IMS calls
- A loop in CICS or IMS code

- A loop in IBM MQ for z/OS

Symptoms of waits and loops

Any of the following symptoms could be caused by a wait, a loop, or by a badly tuned or overloaded system:
- Timeouts on MQGET WAITs
- Batch jobs suspended
- TSO session suspended
- CICS task suspended
- Transactions not being started because of resource constraints, for example CICS MAX task
- Queues becoming full, and not being processed
- System commands not accepted, or producing no response

Related concepts:

"Dealing with waits"
Waits can occur in batch or TSO applications, CICS transactions, and other components. Use this topic to determine where waits can occur.

"Dealing with loops" on page 1760
Loops can occur in different areas of a z/OS system. Use this topic to help determine where a loop is occurring.

Dealing with waits:

Waits can occur in batch or TSO applications, CICS transactions, and other components. Use this topic to determine where waits can occur.

When investigating what appears to be a problem with tasks or subsystems waiting, it is necessary to take into account the environment in which the task or subsystem is running.

It might be that your z/OS system is generally under stress. In this case, there can be many symptoms. If there is not enough real storage, jobs experience waits at paging interrupts or swap-outs. Input/output (I/O) contention or high channel usage can also cause waits.

You can use standard monitoring tools, such as *Resource Monitoring Facility* (RMF) to diagnose such problems. Use normal z/OS tuning techniques to resolve them.

Is a batch or TSO program waiting?

Consider the following points:

Your program might be waiting on another resource
For example, a VSAM control interval (CI) that another program is holding for update.

Your program might be waiting for a message that has not yet arrived
This condition might be normal behavior if, for example, it is a server program that constantly monitors a queue.

Alternatively, your program might be waiting for a message that has arrived, but has not yet been committed.

Issue the DIS CONN(*) TYPE(HANDLE) command and examine the queues in use by your program.

If you suspect that your program has issued an MQI call that did not involve an MQGET WAIT, and control has not returned from IBM MQ, take an SVC dump of both the batch or TSO job, and the IBM MQ subsystem before canceling the batch or TSO program.

Also consider that the wait state might be the result of a problem with another program, such as an abnormal termination (see “Messages do not arrive when expected” on page 1763), or in IBM MQ itself (see “Is IBM MQ waiting for z/OS ?” on page 1760). Refer to “IBM MQ for z/OS dumps” on page 1735 (specifically Figure 128 on page 1737) for information about obtaining a dump.

If the problem persists, refer to “Searching the IBM database for similar problems, and solutions” on page 1639 and “Contacting IBM Software Support” on page 1658 for information about reporting the problem to IBM.

Is a CICS transaction waiting?

Consider the following points:

CICS might be under stress
This might indicate that the maximum number of tasks allowed (MAXTASK) has been reached, or a short on storage (SOS) condition exists. Check the console log for messages that might explain this (for example, SOS messages), or see the *CICS Problem Determination Guide*.

The transaction might be waiting for another resource
For example, this might be file I/O. You can use CEMT INQ TASK to see what the task is waiting for. If the resource type is MQSERIES your transaction is waiting on IBM MQ (either in an **MQGET** WAIT or a task switch). Otherwise see the *CICS Problem Determination Guide* to determine the reason for the wait.

The transaction might be waiting for IBM MQ for z/OS
This might be normal, for example, if your program is a server program that waits for messages to arrive on a queue. Otherwise it might be the result of a transaction abend, for example (see “Messages do not arrive when expected” on page 1763). If so, the abend is reported in the CSMT log.

The transaction might be waiting for a remote message
If you are using distributed queuing, the program might be waiting for a message that has not yet been delivered from a remote system (for further information, refer to “Problems with missing messages when using distributed queuing” on page 1764).

If you suspect that your program has issued an MQI call that did not involve an **MQGET** WAIT (that is, it is in a task switch), and control has not returned from IBM MQ, take an SVC dump of both the CICS region, and the IBM MQ subsystem before canceling the CICS transaction. Refer to “Dealing with loops” on page 1760 for information about waits. Refer to “IBM MQ for z/OS dumps” on page 1735 (specifically Figure 128 on page 1737) for information about obtaining a dump.

If the problem persists, refer to “Searching the IBM database for similar problems, and solutions” on page 1639 and “Contacting IBM Software Support” on page 1658 for information about reporting the problem to IBM.

Is Db2 waiting?

If your investigations indicate that Db2 is waiting, check the following:

1. Use the Db2 -DISPLAY THREAD(*) command to determine if any activity is taking place between the queue manager and the Db2 subsystem.
2. Try and determine whether any waits are local to the queue manager subsystems or are across the Db2 subsystems.

Is RRS active?

- Use the D RRS command to determine if RRS is active.

Is IBM MQ waiting for z/OS ?

If your investigations indicate that IBM MQ itself is waiting, check the following:

1. Use the DISPLAY THREAD(*) command to check if anything is connected to IBM MQ.
2. Use SDSF DA, or the z/OS command `DISPLAY A,xxxxMSTR` to determine whether there is any processor usage (as shown in "Has your application or IBM MQ for z/OS stopped processing work?" on page 1302).
   - If IBM MQ is using some processor time, reconsider other reasons why IBM MQ might be waiting, or consider whether this is actually a performance problem.
   - If there is no processor activity, check whether IBM MQ responds to commands. If you can get a response, reconsider other reasons why IBM MQ might be waiting.
   - If you cannot get a response, check the console log for messages that might explain the wait (for example, IBM MQ might have run out of active log data sets, and be waiting for offload processing).

If you are satisfied that IBM MQ has stalled, use the STOP QMGR command in both QUIESCE and FORCE mode to terminate any programs currently being executed.

If the STOP QMGR command fails to respond, cancel the queue manager with a dump, and restart. If the problem recurs, refer to "Contacting IBM Software Support" on page 1658 for further guidance.

Related concepts:

"Distinguishing between waits and loops" on page 1757
Waits and loops on IBM MQ can present similar symptoms. Use this topic to help determine if you are experiencing a wait of a loop.

"Dealing with loops"
Loops can occur in different areas of a z/OS system. Use this topic to help determine where a loop is occurring.

Dealing with loops:

Loops can occur in different areas of a z/OS system. Use this topic to help determine where a loop is occurring.

The following topics describe the various types of loop that you might encounter, and suggest some responses.

Is a batch application looping?

If you suspect that a batch or TSO application is looping, use the console to issue the z/OS command `DISPLAY JOBS,A` (for a batch application) or `DISPLAY TS,A` (for a TSO application). Note the CT values from the data displayed, and repeat the command.

If any task shows a significant increase in the CT value, it might be that the task is looping. You could also use SDSF DA, which shows you the percentage of processor that each address space is using.

Is a batch job producing a large amount of output?

An example of this behavior might be an application that browses a queue and prints the messages. If the browse operation has been started with BROWSE FIRST, and subsequent calls have not been reset to BROWSE NEXT, the application browses, and prints the first message on the queue repeatedly.

You can use SDSF DA to look at the output of running jobs if you suspect that it might be causing a problem.

Does a CICS region show heavy processor activity?

It might be that a CICS application is looping, or that the CICS region itself is in a loop. You might see AICA abends if a transaction goes into a tight (unyielding) loop.

If you suspect that CICS, or a CICS application is looping, see the *CICS Problem Determination Guide*.

Does an IMS region show heavy processor activity?

It might be that an IMS application is looping. If you suspect this behavior, see *IMS Diagnosis Guide and Reference* l.

Is the queue manager showing heavy processor activity?

Try to enter an MQSC DISPLAY command from the console. If you get no response, it is possible that the queue manager is looping. Follow the procedure shown in "Has your application or IBM MQ for z/OS stopped processing work?" on page 1302 to display information about the processor time being used by the queue manager. If this command indicates that the queue manager is in a loop, take a memory dump, cancel the queue manager and restart.

If the problem persists, see "Searching the IBM database for similar problems, and solutions" on page 1639 and "Contacting IBM Software Support" on page 1658 for information about reporting the problem to IBM.

Is a queue, page set, or Coupling Facility structure filling up unexpectedly?

If so, it might indicate that an application is looping, and putting messages on to a queue. (It might be a batch, CICS, or TSO application.)

Identifying a looping application

In a busy system, it might be difficult to identify which application is causing the problem. If you keep a cross-reference of applications to queues, terminate any programs or transactions that might be putting messages on to the queue. Investigate these programs or transactions before using them again. (The most likely culprits are new, or changed applications; check your change log to identify them.)

Try issuing a DISPLAY QSTATUS command on the queue. This command returns information about the queue that might help to identify which application is looping.

Incorrect triggering definitions

It might be that a getting application has not been triggered because of incorrect object definitions, for example, the queue might be set to NOTRIGGER.

Distributed queuing

Using distributed queuing, a symptom of this problem might be a message in the receiving system indicating that **MQPUT** calls to the dead-letter queue are failing. This problem might be caused because the dead-letter queue has also filled up. The dead-letter queue message header (dead-letter header structure) contains a reason or feedback code explaining why the message might not be put on to the target queue. See MQDLH - Dead-letter header for information about the dead-letter header structure.

Allocation of queues to page sets

If a particular page set frequently fills up, there might be a problem with the allocation of queues to page sets. See IBM MQ for z/OS performance constraints for more information.

Shared queues

Is the Coupling Facility structure full? The z/OS command DISPLAY CF displays information about Coupling Facility storage including the total amount, the total in use, and the total free control and non-control storage. The RMF Coupling Facility Usage Summary Report provides a more permanent copy of this information.

Are a task, and IBM MQ for z/OS, showing heavy processor activity?

In this case, a task might be looping on MQI calls (for example, browsing the same message repeatedly).

Related concepts:

"Distinguishing between waits and loops" on page 1757
Waits and loops on IBM MQ can present similar symptoms. Use this topic to help determine if you are experiencing a wait of a loop.

"Dealing with waits" on page 1758
Waits can occur in batch or TSO applications, CICS transactions, and other components. Use this topic to determine where waits can occur.

Dealing with incorrect output

Incorrect output can be missing, unexpected, or corrupted information. Read this topic to investigate further.

The term "incorrect output⌂ can be interpreted in many different ways, and its meaning for problem determination with this product documentation is explained in "Have you obtained incorrect output?" on page 1311.

The following topics contains information about the problems that you could encounter with your system and classify as incorrect output:

- Application messages that do not arrive when you are expecting them
- Application messages that contain the wrong information, or information that has been corrupted

Additional problems that you might encounter if your application uses distributed queues are also described.

- "Messages do not arrive when expected" on page 1763
- "Problems with missing messages when using distributed queuing" on page 1764
- "Problems with getting messages when using message grouping" on page 1766
- "Finding messages sent to a cluster queue" on page 1766
- "Finding messages sent to the IBM MQ - IMS bridge" on page 1766
- "Messages contain unexpected or corrupted information" on page 1767

Related concepts:

"Making initial checks" on page 1277
There are some initial checks that you can make that may provide answers to common problems that you may have.

"Dealing with performance problems on z/OS" on page 1755
Use this topic to investigate performance problems in more detail.

Messages do not arrive when expected

Missing messages can have different causes. Use this topic to investigate the causes further.

If messages do not arrive on the queue when you are expecting them, check for the following:

Has the message been put onto the queue successfully?

Did IBM MQ issue a return and reason code for the MQPUT, for example:

- Has the queue been defined correctly, for example is MAXMSGL large enough? (reason code 2030).
- Can applications put messages on to the queue (is the queue enabled for MQPUT calls)? (reason code 2051).
- Is the queue already full? This could mean that an application could not put the required message on to the queue (reason code 2053).

Is the queue a shared queue?

- Have Coupling Facility structures been defined successfully in the CFRM policy data set? Messages held on shared queues are stored inside a Coupling Facility.
- Have you activated the CFRM policy?

Is the queue a cluster queue?

If it is, there might be multiple instances of the queue on different queue managers. This means that the messages could be on a different queue manager.

- Did you want the message to go to a cluster queue?
- Is your application designed to work with cluster queues?
- Did the message get put to a different instance of the queue from that expected?

Check any cluster-workload exit programs to see that they are processing messages as intended.

Do your gets fail?

- Does the application need to take a syncpoint?

  If messages are being put or got within syncpoint, they are not available to other tasks until the unit of recovery has been committed.
- Is the time interval on the MQGET long enough?

  If you are using distributed processing, you should allow for reasonable network delays, or problems at the remote end.
- Was the message you are expecting defined as persistent?

  If not, and the queue manager has been restarted, the message will have been deleted. Shared queues are an exception because nonpersistent messages survive a queue manager restart.
- Are you waiting for a specific message that is identified by a message or correlation identifier (`MsgId` or `CorrelId`)?

  Check that you are waiting for a message with the correct `MsgId` or `CorrelId`. A successful MQGET call sets both these values to that of the message got, so you might need to reset these values to get another message successfully.

  Also check if you can get other messages from the queue.
- Can other applications get messages from the queue?

If so, has another application already retrieved the message?

If the queue is a shared queue, check that applications on other queue managers are not getting the messages.

If you cannot find anything wrong with the queue, and the queue manager itself is running, make the following checks on the process that you expected to put the message on to the queue:

- Did the application get started?

  If it should have been triggered, check that the correct trigger options were specified.
- Is a trigger monitor running?
- Was the trigger process defined correctly (both to IBM MQ for z/OS and CICS or IMS)?
- Did it complete correctly?

  Look for evidence of an abend, for example, in the CICS log.
- Did the application commit its changes, or were they backed out?

  Look for messages in the CICS log indicating this.

If multiple transactions are serving the queue, they might occasionally conflict with one another. For example, one transaction might issue an MQGET call with a buffer length of zero to find out the length of the message, and then issue a specific MQGET call specifying the `MsgId` of that message. However, while this is happening, another transaction might have issued a successful MQGET call for that message, so the first application receives a completion code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multi-server environment must be designed to cope with this situation.

Have any of your systems suffered an outage? For example, if the message you were expecting should have been put on to the queue by a CICS application, and the CICS system went down, the message might be in doubt. This means that the queue manager does not know whether the message should be committed or backed out, and so has locked it until this is resolved when resynchronization takes place.

Note: The message is deleted after resynchronization if CICS decides to back it out.

Also consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If so, refer to "Messages contain unexpected or corrupted information" on page 1767.

Problems with missing messages when using distributed queuing

Use this topic tom understand possible causes of missing messages when using distributed queuing.

If your application uses distributed queuing, consider the following points:

Has distributed queuing been correctly installed on both the sending and receiving systems?
: Ensure that the instructions about installing the distributed queue management facility in Configuring z/OS have been followed correctly.

Are the links available between the two systems?
: Check that both systems are available, and connected to IBM MQ for z/OS. Check that the LU 6.2 or TCP/IP connection between the two systems is active or check the connection definitions on any other systems that you are communicating with.

  See Monitoring and performance for more information about trace-route messaging in a network.

Is the channel running?

- Issue the following command for the transmission queue:

  ```
  DISPLAY QUEUE (qname) IPPROCS
  ```

  If the value for IPPROCS is 0, this means that the channel serving this transmission queue is not running.
- Issue the following command for the channel:

```
DISPLAY CHSTATUS (channel-name) STATUS MSGS
```

Use the output produced by this command to check that the channel is serving the correct transmission queue and that it is connected to the correct target machine and port. You can determine whether the channel is running from the STATUS field. You can also see if any messages have been sent on the channel by examining the MSGS field.

If the channel is in RETRYING state, this is probably caused by a problem at the other end. Check that the channel initiator and listener have been started, and that the channel has not been stopped. If somebody has stopped the channel, you need to start it manually.

Is triggering set on in the sending system?
Check that the channel initiator is running.

Does the transmission queue have triggering set on?
If a channel is stopped under specific circumstances, triggering can be set off for the transmission queue.

Is the message you are waiting for a reply message from a remote system?
Check the definitions of the remote system, as previously described, and check that triggering is activated in the remote system. Also check that the LU 6.2 connection between the two systems is not single session (if it is, you cannot receive reply messages).

Check that the queue on the remote queue manager exists, is not full, and accepts the message length. If any of these criteria are not fulfilled, the remote queue manager tries to put the message on the dead-letter queue. If the message length is longer than the maximum length that the channel permits, the sending queue manager tries to put the message on its dead-letter queue.

Is the queue already full?

This could mean that an application could not put the required message on to the queue. If this is so, check if the message has been put on to the dead-letter queue.

The dead-letter queue message header (dead-letter header structure) contains a reason or feedback code explaining why the message could not be put on to the target queue. See MQDLH - Dead-letter header for more information about the dead-letter header structure.

Is there a mismatch between the sending and receiving queue managers?
For example, the message length could be longer than the receiving queue manager can handle. Check the console log for error messages.

Are the channel definitions of the sending and receiving channels compatible?
For example, a mismatch in the wrap value of the sequence number stops the channel. See Distributed queuing and clusters.

Has data conversion been performed correctly?
If a message has come from a different queue manager, are the CCSIDs and encoding the same, or does data conversion need to be performed.

Has your channel been defined for fast delivery of nonpersistent messages?
If your channel has been defined with the NPMSPEED attribute set to FAST (the default), and the channel has stopped for some reason and then been restarted, nonpersistent messages might have been lost. See Nonpersistent message speed (NPMSPEED) for more information about fast messages.

Is a channel exit causing the messages to be processed in an unexpected way?
For example, a security exit might prevent a channel from starting, or an *ExitResponse* of MQXCC_CLOSE_CHANNEL might terminate a channel.

Problems with getting messages when using message grouping

Use this topic to understand some of the issues with getting messages when using message grouping.

Is the application waiting for a complete group of messages?

Ensure all the messages in the group are on the queue. If you are using distributed queuing, see "Problems with missing messages when using distributed queuing" on page 1764. Ensure the last message in the group has the appropriate MsgFlags set in the message descriptor to indicate that it is the last message. Ensure the message expiry of the messages in the group is set to a long enough interval that they do not expire before they are retrieved.

If messages from the group have already been retrieved, and the get request is not in logical order, turn off the option to wait for a complete group when retrieving the other group messages.

If the application issues a get request in logical order for a complete group, and midway through retrieving the group it cannot find a message:

Ensure that no other applications are running against the queue and getting messages. Ensure that the message expiry of the messages in the group is set to a long enough interval that they do not expire before they are retrieved. Ensure that no one has issued the CLEAR QUEUE command. You can retrieve incomplete groups from a queue by getting the messages by group ID, without specifying the logical order option.

Finding messages sent to a cluster queue

Use this topic to understand some of the issues involved with finding messages sent to a cluster queue.

Before you can use the techniques described in these topics to find a message that did not arrive at a cluster queue, you need to determine the queue managers that host the queue to which the message was sent. You can determine this in the following ways:

- You can use the DISPLAY QUEUE command to request information about cluster queues.
- You can use the name of the queue and queue manager that is returned in the MQPMO structure.

  If you specified the MQOO_BIND_ON_OPEN option for the message, these fields give the destination of the message. If the message was not bound to a particular queue and queue manager, these fields give the name of the first queue and queue manager to which the message was sent. In this case, it might not be the ultimate destination of the message.

Finding messages sent to the IBM MQ - IMS bridge

Use this topic to understand possible causes for missing messages sent to the IBM MQ - IMS bridge.

If you are using the IBM MQ - IMS bridge, and your message has not arrived as expected, consider the following:

Is the IBM MQ - IMS bridge running?

Issue the following command for the bridge queue:

```
DISPLAY QSTATUS(qname) IPPROCS CURDEPTH
```

The value of IPPROCS should be 1; if it is 0, check the following:

- Is the queue a bridge queue?
- Is IMS running?
- Has OTMA been started?
- Is IBM MQ connected to OTMA?

  Note: There are two IBM MQ messages that you can use to establish whether you have a connection to OTMA. If message CSQ2010I is present in the job log of the task, but message CSQ2011I is not present, IBM MQ is connected to OTMA. This message also tells you to which IBM MQ system OTMA is connected. For more information about the content of these messages, see IBM MQ for z/OS messages, completion, and reason codes.

Within the queue manager there is a task processing each IMS bridge queue. This task gets from the queue, sends the request to IMS, and then does a commit. If persistent messages are used, then the commit requires disk I/O and so the process takes longer than for non-persistent messages. The time to process the get, send, and commit, limits the rate at which the task can process messages. If the task can keep up with the workload then the current depth is close to zero. If you find that the current depth is often greater than zero you might be able to increase throughput by using two queues instead of one.

Use the IMS command `/DIS OTMA` to check that OTMA is active.

If your messages are flowing to IMS, check the following:

- Use the IMS command `/DIS TMEMBER client TPIPE ALL` to display information about IMS Tpipes. From this you can determine the number of messages enqueued on, and dequeued from, each Tpipe. (Commit mode 1 messages are not usually queued on a Tpipe.)
- Use the IMS command `/DIS A` to show whether there is a dependent region available for the IMS transaction to run in.
- Use the IMS command `/DIS TRAN trancode` to show the number of messages queued for a transaction.
- Use the IMS command `/DIS PROG progname` to show if a program has been stopped.

Was the reply message sent to the correct place?

Issue the following command:

```
DISPLAY QSTATUS(*) CURDEPTH
```

Does the CURDEPTH indicate that there is a reply on a queue that you are not expecting?

Messages contain unexpected or corrupted information

Use this topic to understand some of the issues that can cause unexpected or corrupted output.

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following points:

Has your application, or the application that put the message on to the queue changed?

Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

For example, a copybook formatting the message might have been changed, in which case, both applications have to be recompiled to pick up the changes. If one application has not been recompiled, the data will appear corrupt to the other.

Check that no external source of data, such as a VSAM data set, has changed. This could also invalidate your data if any necessary recompilations have not been done. Also check that any CICS maps and TSO panels that you are using for input of message data have not changed.

Is an application sending messages to the wrong queue?

Check that the messages your application is receiving are not intended for an application servicing a different queue. If necessary, change your security definitions to prevent unauthorized applications from putting messages on to the wrong queues.

If your application has used an alias queue, check that the alias points to the correct queue.

If you altered the queue to make it a cluster queue, it might now contain messages from different application sources.

Has the trigger information been specified correctly for this queue?

Check that your application should have been started, or should a different application have been started?

Has data conversion been performed correctly?

If a message has come from a different queue manager, are the CCSIDs and encoding the same, or does data conversion need to be performed.

Check that the `Format` field of the MQMD structure corresponds with the content of the message. If not, the data conversion process might not have been able to deal with the message correctly.

If these checks do not enable you to solve the problem, check your application logic, both for the program sending the message, and for the program receiving it.

Dealing with issues when capturing SMF data for the channel initiator (CHINIT)

Channel accounting and CHINIT statistics SMF data might not be captured for various reasons.

For more information, see:

Related information:

Layout of SMF records for the channel initiator

Troubleshooting channel accounting data

Checks to carry out if channel accounting SMF data is not being produced for channels.

Procedure

1. Check that you have STATCHL set, either at the queue manager or the channel level.
   - A value of OFF at channel level means that data is not collected for this channel.
   - A value of OFF at queue manager level means data is not collected for channels with STATCHL(QMGR).
   - A value of NONE (only applicable at queue manager level) means data is not collected for all channels, regardless of their STATCHL setting.
2. For client channels check that STATCHL is set at the queue manager level.
3. For automatically defined cluster sender channels, check that the STATACLS is set.
4. Issue the display trace command. You need TRACE(A) CLASS(4) enabled for channel accounting data to be collected.
5. If the trace is enabled, SMF data is written:
   - On a timed interval, depending on the value of the STATIME system parameter. A value of zero means that the SMF statistics broadcast is used. Use the DIS SYSTEM command to display the value of STATIME.
   - If the SET SYSTEM command is issued to change the value of the STATIME system parameter.
   - When the CHINIT is shut down.
   - If the STOP TRACE(A) CLASS(4) is issued, any accounting data is written out.
6. SMF might hold the data in memory before writing it out to the SMF data sets or the SMF structure. Issue the MVS command **`D SMF,O`** and note the MAXDORM value. SMF can keep the data in memory for the MAXDORM period before writing it out.

Related information:

Planning for channel initiator SMF data

Interpreting IBM MQ performance statistics

Troubleshooting CHINIT statistics data

Checks to carry out if CHINIT statistics SMF data is not being produced.

Procedure

1. Issue the display trace command. You need TRACE(S) CLASS(4) enabled for information about the CHINIT.
2. If the trace is enabled, SMF data is written:
   - On a timed interval, depending on the value of the STATIME system parameter. A value of zero means that the SMF statistics broadcast is used. Use the DIS SYSTEM command to display the value of STATIME.
   - If the SET SYSTEM command is issued to change the value of the STATIME system parameter.
   - When the CHINIT is shut down.
   - If the STOP TRACE(S) CLASS(4) is issued, any statistics data is written out.
3. SMF can hold the data in memory before writing it out to the SMF data sets or the SMF structure. Issue the MVS command **`D SMF,O`** and note the MAXDORM value. SMF can keep the data in memory for the MAXDORM period before writing it out.

Problem determination in DQM

Aspects of problem determination relating to distributed queue management (DQM) and suggested methods of resolving problems.

Some of the problems that are described are platform and installation specific. Where this is the case, it is made clear in the text.

IBM MQ provides a utility to assist with problem determination named **`amqldmpa`**. During the course of problem determination, your IBM service representative might ask you to provide output from the utility.

Your IBM service representative will provide you with the parameters you require to collect the appropriate diagnostic information, and information on how you send the data you record to IBM.

Attention: You should not rely on the format of the output from this utility, as the format is subject to change without notice.

Problem determination for the following scenarios is discussed:

- "Error message from channel control" on page 1770
- "Ping" on page 1771
- "Dead-letter queue considerations" on page 1771
- "Validation checks" on page 1772
- "In-doubt relationship" on page 1772
- "Channel startup negotiation errors" on page 1772
- "When a channel refuses to run" on page 1773
- "Retrying the link" on page 1775
- "Data structures" on page 1776
- "User exit problems" on page 1776
- "Disaster recovery" on page 1776
- "Channel switching" on page 1777

- “Connection switching” on page 1777
- “Client problems” on page 1777
- “Error logs” on page 1778
- “Message monitoring” on page 1779

Related concepts:

“Troubleshooting and support” on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

“Making initial checks on Windows, UNIX and Linux systems” on page 1277
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

z/OS “Making initial checks on z/OS” on page 1297
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

IBM i “Making initial checks on IBM i” on page 1288
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

“Reason codes and exceptions” on page 1314
You can use the following messages and reason codes to help you solve problems with your IBM MQ components or applications.

Related information:

Configuring distributed queuing

Communications protocol return codes

Error message from channel control

Problems found during normal operation of the channels are reported to the system console and to the system log. In IBM MQ for Windows they are reported to the channel log. Problem diagnosis starts with the collection of all relevant information from the log, and analysis of this information to identify the problem.

However, this could be difficult in a network where the problem may arise at an intermediate system that is staging some of your messages. An error situation, such as transmission queue full, followed by the dead-letter queue filling up, would result in your channel to that site closing down.

In this example, the error message you receive in your error log will indicate a problem originating from the remote site, but may not be able to tell you any details about the error at that site.

You need to contact your counterpart at the remote site to obtain details of the problem, and to receive notification of that channel becoming available again.

Ping

Ping is useful in determining whether the communication link and the two message channel agents that make up a message channel are functioning across all interfaces.

Ping makes no use of transmission queues, but it does invoke some user exit programs. If any error conditions are encountered, error messages are issued.

To use ping, you can issue the MQSC command PING CHANNEL. On z/OS z/OS IBM i and i5/OS , you can also use the panel interface to select this option.

On UNIX platforms, IBM i i5/OS, and Windows, you can also use the MQSC command PING QMGR to test whether the queue manager is responsive to commands.

Dead-letter queue considerations

In some IBM MQ implementations the dead-letter queue is referred to as an *undelivered-message queue*.

If a channel ceases to run for any reason, applications will probably continue to place messages on transmission queues, creating a potential overflow situation. Applications can monitor transmission queues to find the number of messages waiting to be sent, but this would not be a normal function for them to carry out.

When this occurs in a message-originating node, and the local transmission queue is full, the application's PUT fails.

When this occurs in a staging or destination node, there are three ways that the MCA copes with the situation:

1. By calling the message-retry exit, if one is defined.
2. By directing all overflow messages to a *dead-letter queue* (DLQ), returning an exception report to applications that requested these reports.

   Note: In distributed-queuing management, if the message is too big for the DLQ, the DLQ is full, or the DLQ is not available, the channel stops and the message remains on the transmission queue. Ensure your DLQ is defined, available, and sized for the largest messages you handle.
3. By closing down the channel, if neither of the previous options succeeded.
4. By returning the undelivered messages back to the sending end and returning a full report to the reply-to queue (MQRC_EXCEPTION_WITH_FULL_DATA and MQRO_DISCARD_MSG).

If an MCA is unable to put a message on the DLQ:

- The channel stops
- Appropriate error messages are issued at the system consoles at both ends of the message channel
- The unit of work is backed out, and the messages reappear on the transmission queue at the sending channel end of the channel
- Triggering is disabled for the transmission queue

Validation checks

A number of validation checks are made when creating, altering, and deleting channels, and where appropriate, an error message returned.

Errors may occur when:

- A duplicate channel name is chosen when creating a channel
- Unacceptable data is entered in the channel parameter fields
- The channel to be altered is in doubt, or does not exist

In-doubt relationship

If a channel is in doubt, it is usually resolved automatically on restart, so the system operator does not need to resolve a channel manually in normal circumstances. See In-doubt channels for further information.

Channel startup negotiation errors

During channel startup, the starting end has to state its position and agree channel running parameters with the corresponding channel. It may happen that the two ends cannot agree on the parameters, in which case the channel closes down with error messages being issued to the appropriate error logs.

Shared channel recovery

The following table shows the types of shared-channel failure and how each type is handled.

Type of failure:	**What happens:**
Channel initiator communications subsystem failure	The channels dependent on the communications subsystem enter channel retry, and are restarted on an appropriate queue-sharing group channel initiator by a load-balanced start command.
Channel initiator failure	The channel initiator fails, but the associated queue manager remains active. The queue manager monitors the failure and initiates recovery processing.
Queue manager failure	The queue manager fails (failing the associated channel initiator). Other queue managers in the queue-sharing group monitor the event and initiate peer recovery.
Shared status failure	Channel state information is stored in Db2, so a loss of connectivity to Db2 becomes a failure when a channel state change occurs. Running channels can carry on running without access to these resources. On a failed access to Db2, the channel enters retry.

Shared channel recovery processing on behalf of a failed system requires connectivity to Db2 to be available on the system managing the recovery to retrieve the shared channel status.

When a channel refuses to run

If a channel refuses to run, there are a number of potential reasons.

Carry out the following checks:

- Check that DQM and the channels have been set up correctly. This is a likely problem source if the channel has never run. Reasons could be:
  - A mismatch of names between sending and receiving channels (remember that uppercase and lowercase letters are significant)
  - Incorrect channel types specified
  - The sequence number queue (if applicable) is not available, or is damaged
  - The dead-letter queue is not available
  - The sequence number wrap value is different on the two channel definitions
  - A queue manager or communication link is not available
  - A receiver channel might be in STOPPED state
  - The connection might not be defined correctly
  - There might be a problem with the communications software (for example, is TCP running?)
- It is possible that an in-doubt situation exists, if the automatic synchronization on startup has failed for some reason. This is indicated by messages on the system console, and the status panel may be used to show channels that are in doubt.

  The possible responses to this situation are:
  - Issue a Resolve channel request with Backout or Commit.

    You need to check with your remote link supervisor to establish the number of the last-committed unit of work ID (LUWID) committed. Check this against the last number at your end of the link. If the remote end has committed a number, and that number is not yet committed at your end of the link, then issue a RESOLVE COMMIT command.

    In all other cases, issue a RESOLVE BACKOUT command.

    The effect of these commands is that backed out messages reappear on the transmission queue and are sent again, while committed messages are discarded.

    If in doubt yourself, perhaps backing out with the probability of duplicating a sent message would be the safer decision.
  - Issue a RESET CHANNEL command.

    This command is for use when sequential numbering is in effect, and should be used with care. Its purpose is to reset the sequence number of messages and you should use it only after using the RESOLVE command to resolve any in-doubt situations.
- Windows UNIX IBM i z/OS When sequential numbering is being used, and a sender channel starts up after being reset, the sender channel takes two actions:
  - It tells the receiver channel that it has been reset.
  - It specifies the next message sequence number that is to be used by both the sender and receiver channels.
- If the status of a receiver end of the channel is STOPPED, it can be reset by starting the receiver end.

  Note: This does not start the channel, it merely resets the status. The channel must still be started from the sender end.

Triggered channels

If a triggered channel refuses to run, investigate the possibility of in-doubt messages here: "When a channel refuses to run" on page 1773

Another possibility is that the trigger control parameter on the transmission queue has been set to NOTRIGGER by the channel. This happens when:

- There is a channel error.
- The channel was stopped because of a request from the receiver.
- The channel was stopped because of a problem on the sender that requires manual intervention.

After diagnosing and fixing the problem, start the channel manually.

An example of a situation where a triggered channel fails to start is as follows:

1. A transmission queue is defined with a trigger type of FIRST.
2. A message arrives on the transmission queue, and a trigger message is produced.
3. The channel is started, but stops immediately because the communications to the remote system are not available.
4. The remote system is made available.
5. Another message arrives on the transmission queue.
6. The second message does not increase the queue depth from zero to one, so no trigger message is produced (unless the channel is in RETRY state). If this happens, restart the channel manually.

On IBM MQ for z/OS, if the queue manager is stopped using MODE(FORCE) during channel initiator shutdown, it might be necessary to manually restart some channels after channel initiator restart.

Conversion failure

Another reason for the channel refusing to run could be that neither end is able to carry out necessary conversion of message descriptor data between ASCII and EBCDIC, and integer formats. In this instance, communication is not possible.

Network problems

There are a number of things to check if you are experiencing network problems.

When using LU 6.2, make sure that your definitions are consistent throughout the network. For example, if you have increased the RU sizes in your CICS Transaction Server for z/OS or Communications Manager definitions, but you have a controller with a small MAXDATA value in its definition, the session might fail if you attempt to send large messages across the network. A symptom of this problem might be that channel negotiation takes place successfully, but the link fails when message transfer occurs.

When using TCP, if your channels are unreliable and your connections break, you can set a KEEPALIVE value for your system or channels. You do this using the SO_KEEPALIVE option to set a system-wide value.

z/OS On IBM MQ for z/OS, you also have the following options:

- Use the Keepalive Interval channel attribute (KAINT) to set channel-specific keepalive values.
- Use the RCVTIME and RCVTMIN channel initiator parameters.

These options are discussed in Checking that the other end of the channel is still available, and Keepalive Interval (KAINT).

Adopting an MCA

The Adopt MCA function enables IBM MQ to cancel a receiver channel and to start a new one in its place.

For more information about this function, see Adopting an MCA.

Registration time for DDNS

When a group TCP/IP listener is started, it registers with DDNS. But there can be a delay until the address is available to the network. A channel that is started in this period, and which targets the newly registered generic name, fails with an 'error in communications configuration' message. The channel then goes into retry until the name becomes available to the network. The length of the delay is dependent on the name server configuration used.

Dial-up problems

IBM MQ supports connection over dial-up lines but you should be aware that with TCP, some protocol providers assign a new IP address each time you dial in. This can cause channel synchronization problems because the channel cannot recognize the new IP addresses and so cannot ensure the authenticity of the partner. If you encounter this problem, you need to use a security exit program to override the connection name for the session.

This problem does not occur when an IBM MQ for IBM i, UNIX systems, or Windows systems product is communicating with another product at the same level, because the queue manager name is used for synchronization instead of the IP address.

Retrying the link

An error scenario may occur that is difficult to recognize. For example, the link and channel may be functioning perfectly, but some occurrence at the receiving end causes the receiver to stop. Another unforeseen situation could be that the receiver system has run out of memory and is unable to complete a transaction.

You need to be aware that such situations can arise, often characterized by a system that appears to be busy but is not actually moving messages. You need to work with your counterpart at the far end of the link to help detect the problem and correct it.

Retry considerations

If a link failure occurs during normal operation, a sender or server channel program will itself start another instance, provided that:
1. Initial data negotiation and security exchanges are complete
2. The retry count in the channel definition is greater than zero

Note: For IBM i, UNIX systems, and Windows, to attempt a retry a channel initiator must be running. In platforms other than IBM MQ for IBM i, UNIX systems, and Windows systems, this channel initiator must be monitoring the initiation queue specified in the transmission queue that the channel is using.

Shared channel recovery on z/OS:

See “Shared channel recovery” on page 1772, which includes a table that shows the types of shared-channel failure and how each type is handled.

Data structures

Data structures are needed for reference when checking logs and trace entries during problem diagnosis.

More information can be found in Channel-exit calls and data structures and Developing applications reference.

User exit problems

The interaction between the channel programs and the user-exit programs has some error-checking routines, but this facility can only work successfully when the user exits obey certain rules.

These rules are described in Channel-exit programs for messaging channels. When errors occur, the most likely outcome is that the channel stops and the channel program issues an error message, together with any return codes from the user exit. Any errors detected on the user exit side of the interface can be determined by scanning the messages created by the user exit itself.

You might need to use a trace facility of your host system to identify the problem.

Disaster recovery

Disaster recovery planning is the responsibility of individual installations, and the functions performed may include the provision of regular system 'snapshot' dumps that are stored safely off-site. These dumps would be available for regenerating the system, should some disaster overtake it. If this occurs, you need to know what to expect of the messages, and the following description is intended to start you thinking about it.

First a recap on system restart. If a system fails for any reason, it may have a system log that allows the applications running at the time of failure to be regenerated by replaying the system software from a syncpoint forward to the instant of failure. If this occurs without error, the worst that can happen is that message channel syncpoints to the adjacent system may fail on startup, and that the last batches of messages for the various channels will be sent again. Persistent messages will be recovered and sent again, nonpersistent messages may be lost.

If the system has no system log for recovery, or if the system recovery fails, or where the disaster recovery procedure is invoked, the channels and transmission queues may be recovered to an earlier state, and the messages held on local queues at the sending and receiving end of channels may be inconsistent.

Messages may have been lost that were put on local queues. The consequence of this happening depends on the particular IBM MQ implementation, and the channel attributes. For example, where strict message sequencing is in force, the receiving channel detects a sequence number gap, and the channel closes down for manual intervention. Recovery then depends upon application design, as in the worst case the sending application may need to restart from an earlier message sequence number.

Channel switching

A possible solution to the problem of a channel ceasing to run would be to have two message channels defined for the same transmission queue, but with different communication links. One message channel would be preferred, the other would be a replacement for use when the preferred channel is unavailable.

If triggering is required for these message channels, the associated process definitions must exist for each sender channel end.

To switch message channels:
- If the channel is triggered, set the transmission queue attribute NOTRIGGER.
- Ensure the current channel is inactive.
- Resolve any in-doubt messages on the current channel.
- If the channel is triggered, change the process attribute in the transmission queue to name the process associated with the replacement channel.

  In this context, some implementations allow a channel to have a blank process object definition, in which case you may omit this step as the queue manager will find and start the appropriate process object.
- Restart the channel, or if the channel was triggered, set the transmission queue attribute TRIGGER.

Connection switching

Another solution would be to switch communication connections from the transmission queues.

To do this:
- If the sender channel is triggered, set the transmission queue attribute NOTRIGGER.
- Ensure the channel is inactive.
- Change the connection and profile fields to connect to the replacement communication link.
- Ensure that the corresponding channel at the remote end has been defined.
- Restart the channel, or if the sender channel was triggered, set the transmission queue attribute TRIGGER.

Client problems

A client application may receive an unexpected error return code, for example:
- Queue manager not available
- Queue manager name error
- Connection broken

Look in the client error log for a message explaining the cause of the failure. There may also be errors logged at the server, depending on the nature of the failure.

Terminating clients

Even though a client has terminated, it is still possible for its surrogate process to be holding its queues open. Normally this will only be for a short time until the communications layer notifies that the partner has gone.

Error logs

IBM MQ error messages are placed in different error logs depending on the platform. There are error logs for:

- Windows
- UNIX systems
- z/OS

Error logs for Windows

IBM MQ for Windows uses a number of error logs to capture messages concerning the operation of IBM MQ itself, any queue managers that you start, and error data coming from the channels that are in use.

The location the error logs are stored in depends on whether the queue manager name is known and whether the error is associated with a client.

- If the queue manager name is known and the queue manager is available:
  ```
  <install directory>\QMGRS\QMgrName\ERRORS\AMQERR01.LOG
  ```
- If the queue manager is not available:
  ```
  <install directory>\QMGRS\@SYSTEM\ERRORS\AMQERR01.LOG
  ```
- If an error has occurred with a client application:
  ```
  <install directory>\ERRORS\AMQERR01.LOG
  ```

On Windows, you should also examine the Windows application event log for relevant messages.

Error logs on UNIX and Linux systems

IBM MQ on UNIX and Linux systems uses a number of error logs to capture messages concerning the operation of IBM MQ itself, any queue managers that you start, and error data coming from the channels that are in use. The location the error logs are stored in depends on whether the queue manager name is known and whether the error is associated with a client.

- If the queue manager name is known:
  ```
  /var/mqm/qmgrs/QMgrName/errors
  ```
- If the queue manager name is not known (for example when there are problems in the listener or SSL handshake):
  ```
  /var/mqm/errors
  ```

When a client is installed, and there is a problem in the client application, the following log is used:

- If an error has occurred with a client application:
  ```
  /var/mqm/errors/
  ```

Error logs on z/OS

Error messages are written to:
- The z/OS system console
- The channel-initiator job log

If you are using the z/OS message processing facility to suppress messages, the console messages might be suppressed. See Planning your IBM MQ environment on z/OS.

Message monitoring

If a message does not reach its intended destination, you can use the IBM MQ display route application, available through the control command **dspmqrte** , to determine the route a message takes through the queue manager network and its final location.

The IBM MQ display route application is described in the IBM MQ display route application section.

Channel authentication records troubleshooting

If you are having problems using channel authentication records, check whether the problem is described in the following information.

What address are you presenting to the queue manager?

The address that your channel presents to the queue manager depends on the network adapter being used. For example, if the CONNAME you use to get to the listener is "localhost", you present 127.0.0.1 as your address; if it is the real IP address of your computer, then that is the address you present to the queue manager. You might invoke different authentication rules for 127.0.0.1 and your real IP address.

Using BLOCKADDR with channel names

If you use SET CHLAUTH TYPE(BLOCKADDR), it must have the generic channel name CHLAUTH(*) and nothing else. You must block access from the specified addresses using any channel name.

z/OS

CHLAUTH(*) on z/OS systems

On z/OS, a channel name including the asterisk (*) must be enclosed in quotation marks. This rule also applies to the use of a single asterisk to match all channel names. Thus, where you would specify CHLAUTH(*) on other platforms, on z/OS you must specify CHLAUTH('*').

Behavior of SET CHLAUTH command over queue manager restart

If the SYSTEM.CHLAUTH.DATA.QUEUE, has been deleted or altered in a way that it is no longer accessible i.e. PUT(DISABLED), the `SET CHLAUTH` command will only be partially successful. In this instance, `SET CHLAUTH` will update the in-memory cache, but will fail when hardening.

This means that although the rule put in place by the `SET CHLAUTH` command may be operable initially, the effect of the command will not persist over a queue manager restart. The user should investigate, ensuring the queue is accessible and then reissue the command (using `ACTION(REPLACE)`) before cycling the queue manager.

If the SYSTEM.CHLAUTH.DATA.QUEUE remains inaccessible at queue manager startup, the cache of saved rules cannot be loaded and all channels will be blocked until the queue and rules become accessible.

z/OS

Maximum size of ADDRESS and ADDRLIST on z/OS systems

On z/OS, the maximum size for the ADDRESS and ADDRLIST fields are 48 characters. Some IPv6 address patterns could be longer than this limit, for example '0000-ffff:0000-ffff:0000-ffff:0000-ffff:0000-ffff:0000-ffff:0000-ffff:0000-ffff'. In this case, you could use '*' instead.

If you want to use a pattern more than 48 characters long, try to express the requirement in a different way. For example, instead of specifying

'0001-fffe:0001-fffe:0001-fffe:0001-fffe:0001-fffe:0001-fffe:0001-fffe:0001-fffe' as the address pattern for a USERSRC(MAP), you could specify three rules:
- USERSRC(MAP) for all addresses (*)
- USERSRC(NOACCESS) for address '0000:0000:0000:0000:0000:0000:0000:0000'
- USERSRC(NOACCESS) for address 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff'

Commands troubleshooting

- **Scenario:** You receive errors when you use special characters in descriptive text for some commands.
- **Explanation:** Some characters, for example, back slash (\) and double quote (") characters have special meanings when used with commands.
- **Solution:** Precede special characters with a \, that is, enter \\ or \" if you want \ or " in your text. Not all characters are allowed to be used with commands. For more information about characters with special meanings and how to use them, see Characters with special meanings.

Distributed publish/subscribe troubleshooting

Use the advice given in the subtopics to help you to detect and deal with problems when you use publish/subscribe clusters or hierarchies.

Before you begin

If your problems relate to clustering in general, rather than to publish/subscribe messaging using clusters, see“Queue manager clusters troubleshooting” on page 1805.

There are also some helpful troubleshooting tips in Design considerations for retained publications in publish/subscribe clusters.

Related information:

Configuring a publish/subscribe cluster

Designing publish/subscribe clusters

Distributed publish/subscribe system queue errors

Routing for publish/subscribe clusters: Notes on behavior

Use the advice given here to help you to detect and deal with routing problems when you are using clustered publish/subscribe messaging.

For information about status checking and troubleshooting for any queue manager cluster, see “Queue manager clusters troubleshooting” on page 1805.

- All clustered definitions of the same named topic object in a cluster must have the same **CLROUTE** setting. You can check the **CLROUTE** setting for all topics on all hosts in the cluster using the following MQSC command:

  ```
  display tcluster(*) clroute
  ```

- The **CLROUTE** property has no effect unless the topic object specifies a value for the **CLUSTER** property.
- Check that you have spelled the cluster name correctly on your topic. You can define a cluster object such as a topic before defining the cluster. Therefore, when you define a cluster topic, no validation is done on the cluster name because it might not yet exist. Consequently, the product does not alert you to misspelt cluster names.
- When you set the **CLROUTE** property, if the queue manager knows of a clustered definition of the same object from another queue manager that has a different **CLROUTE** setting, the system generates an `MQRCCF_CLUSTER_TOPIC_CONFLICT` exception. However, through near simultaneous object definition on different queue managers, or erratic connectivity with full repositories, differing definitions might be created. In this situation the full repository queue managers arbitrate, accepting one definition and reporting an error for the other one. To get more information about the conflict, use the following MQSC command to check the cluster state of all topics on all queue managers in the cluster:

  ```
  display tcluster(*) clstate
  ```

  A state of `invalid`, or `pending` (if this does not soon turn to active), indicates a problem. If an invalid topic definition is detected, identify the incorrect topic definition and remove it from the cluster. The full repositories have information about which definition was accepted and which was rejected, and the queue managers that created the conflict have some indication of the nature of the problem. See also CLSTATE in DISPLAY TOPIC.
- Setting the **CLROUTE** parameter at a point in the topic tree causes the entire branch beneath it to route topics in that way. You cannot change the routing behavior of a sub-branch of this branch. For this reason, defining a topic object for a lower or higher node in the topic tree with a different **CLROUTE** setting is rejected with an `MQRCCF_CLUSTER_TOPIC_CONFLICT` exception.
- You can use the following MQSC command to check the topic status of all the topics in the topic tree:

  ```
  display tpstatus('#')
  ```

  If you have a large number of branches in the topic tree, the previous command might display status for an inconveniently large number of topics. If that is the case, you can instead display a manageably small branch of the tree, or an individual topic in the tree. The information displayed includes the topic string, cluster name and cluster route setting. It also includes the publisher count and subscription count (number of publishers and subscribers), to help you judge whether the number of users of this topic is as you expect.
- Changing the cluster routing of a topic in a cluster is a significant change to the publish/subscribe topology. After a topic object has been clustered (through setting the **CLUSTER** property) you cannot change the value of the **CLROUTE** property. The object must be un-clustered (**CLUSTER** set to **' '**) before you can change the value. Un-clustering a topic converts the topic definition to a local topic, which results in a period during which publications are not delivered to subscriptions on remote queue managers; this should be considered when performing this change. See The effect of defining a non-cluster topic with the same name as a cluster topic from another queue manager. If you try to change the value of the **CLROUTE** property while it is clustered, the system generates an `MQRCCF_CLROUTE_NOT_ALTERABLE` exception.
- For topic host routing, you can explore alternative routes through the cluster by adding and removing the same cluster topic definition on a range of cluster queue managers. To stop a given queue manager from acting as a topic host for your cluster topic, either delete the topic object, or use the `PUB(DISABLED)` setting to quiesce message traffic for this topic, as discussed in Special handling for the PUB parameter. Do not un-cluster the topic by setting the **CLUSTER** property to **' '**, because removing the cluster name converts the topic definition to a local topic, and prevents the clustering behavior of the topic when used from this queue manager. See The effect of defining a non-cluster topic with the same name as a cluster topic from another queue manager.
- You cannot change the cluster of a sub-branch of the topic tree when the branch has already been clustered to a different cluster and **CLROUTE** is set to `TOPICHOST`. If such a definition is detected at define time, the system generates an `MQRCCF_CLUSTER_TOPIC_CONFLICT` exception. Similarly, inserting a newly

clustered topic definition at a higher node for a different cluster generates an exception. Because of the clustering timing issues previously described, if such an inconsistency is later detected, the queue manager issues errors to the queue manager log.

Related information:

Configuring a publish/subscribe cluster

Designing publish/subscribe clusters

Checking proxy subscription locations

A proxy subscription enables a publication to flow to a subscriber on a remote queue manager. If your subscribers are not getting messages that are published elsewhere in the queue manager network, check that your proxy subscriptions are where you expect them to be.

Missing proxy subscriptions can show that your application is not subscribing on the correct topic object or topic string, or that there is a problem with the topic definition, or that a channel is not running or is not configured correctly.

To show proxy subscriptions, use the following MQSC command:

```
display sub(*) subtype(proxy)
```

Proxy subscriptions are used in all distributed publish/subscribe topologies (hierarchies and clusters). For a topic host routed cluster topic, a proxy subscription exists on each topic hosts for that topic. For a direct routed cluster topic, the proxy subscription exists on every queue manager in the cluster. Proxy subscriptions can also be made to exist on every queue manager in the network by setting the `proxysub(force)` attribute on a topic.

See also Subscription performance in publish/subscribe networks.

Resynchronization of proxy subscriptions

Under normal circumstances, queue managers automatically ensure that the proxy subscriptions in the system correctly reflect the subscriptions on each queue manager in the network. Should the need arise, you can manually resynchronize a queue manager's local subscriptions with the proxy subscriptions that it propagated across the network using the **`REFRESH QMGR TYPE(PROXYSUB)`** command. However, you should do so only in exceptional circumstances.

When to manually resynchronize proxy subscriptions

When a queue manager is receiving subscriptions that it should not be sent, or not receiving subscriptions that it should receive, you should consider manually resynchronizing the proxy subscriptions. However, resynchronization temporarily creates a sudden additional proxy subscription load on the network, originating from the queue manager where the command is issued. For this reason, do not manually resynchronize unless IBM MQ service, IBM MQ documentation, or error logging instructs you to do so.

You do not need to manually resynchronize proxy subscriptions if automatic revalidation by the queue manager is about to occur. Typically, a queue manager revalidates proxy subscriptions with affected directly-connected queue managers at the following times:

- When forming a hierarchical connection
- When modifying the **`PUBSCOPE`** or **`SUBSCOPE`** or **`CLUSTER`** attributes on a topic object
- When restarting the queue manager

Sometimes a configuration error results in missing or extraneous proxy subscriptions:

- Missing proxy subscriptions can be caused if the closest matching topic definition is specified with **Subscription scope** set to `Queue Manager` or with an empty or incorrect cluster name. Note that **Publication scope** does not prevent the sending of proxy subscriptions, but does prevent publications from being delivered to them.
- Extraneous proxy subscriptions can be caused if the closest matching topic definition is specified with **Proxy subscription behavior** set to `Force`.

When configuration errors cause these problems, manual resynchronization does not resolve them. In these cases, amend the configuration.

The following list describes the exceptional situations in which you should manually resynchronize proxy subscriptions:
- After issuing a **REFRESH CLUSTER** command on a queue manager in a publish/subscribe cluster.
- When messages in the queue manager error log tell you to run the **REFRESH QMGR TYPE(REPOS)** command.
- When a queue manager cannot correctly propagate its proxy subscriptions, perhaps because a channel has stopped and all messages cannot be queued for transmission, or because operator error has caused messages to be incorrectly deleted from the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` queue.
- When messages are incorrectly deleted from other system queues.
- When a **DELETE SUB** command is issued in error on a proxy subscription.
- As part of disaster recovery.

How to manually resynchronize proxy subscriptions

First rectify the original problem (for example by restarting the channel), then issue the following command on the queue manager:

```
REFRESH QMGR TYPE(PROXYSUB)
```

When you issue this command, the queue manager sends, to each of its directly-connected queue managers, a list of its own topic strings for which proxy subscriptions should exist. The directly-connected queue managers then update their held proxy subscriptions to match the list. Next, the directly-connected queue managers send back to the originating queue manager a list of their own topic strings for which proxy subscriptions should exist, and the originating queue manager updates its held proxy subscriptions accordingly.

Important usage notes:
- Publications missed due to proxy subscriptions not being in place are not recovered for the affected subscriptions.
- Resynchronization requires the queue manager to start channels to other queue managers. If you are using direct routing in a cluster, or you are using topic host routing and this command is issued on a topic host queue manager, the queue manager will start channels to all other queue managers in the cluster, even those that have not performed publish/subscribe work. Therefore the queue manager that you are refreshing must have enough capability to cope with communicating with every other queue manager in the cluster.
- z/OS If this command is issued on z/OS when the CHINIT is not running, the command is queued up and processed when the CHINIT starts.

Related information:

Checking that async commands for distributed networks have finished

REFRESH CLUSTER considerations for publish/subscribe clusters

Loop detection in a distributed publish/subscribe network

In a distributed publish/subscribe network, it is important that publications and proxy subscriptions cannot loop, because this would result in a flooded network with connected subscribers receiving multiple copies of the same original publication.

The proxy subscription aggregation system described in Proxy subscriptions in a publish/subscribe network does not prevent the formation of a loop, although it will prevent the perpetual looping of proxy subscriptions. Because the propagation of publications is determined by the existence of proxy subscriptions, they can enter a perpetual loop. Websphere MQ V7.0 uses the following technique to prevent publications from perpetually looping:

As publications move around a publish/subscribe topology each queue manager adds a unique fingerprint to the message header. Whenever a publish/subscribe queue manager receives a publication from another publish/subscribe queue manager, the fingerprints held in the message header are checked. If its own fingerprint is already present, the publication has fully circulated around a loop, so the queue manager discards the message, and adds an entry to the error log.

Note: Within a loop, publications are propagated in both directions around the loop, and each queue manager within the loop receives both publications before the originating queue manager discards the looped publications. This results in subscribing applications receiving duplicate copies of publications until the loop is broken.

Loop detection fingerprint format

The loop detection fingerprints are inserted into an RFH2 header or flow as part of the V8.0 protocol. An RFH2 programmer needs to understand the header and pass on the fingerprint information intact. earlier versions of IBM Integration Bus use RFH1 headers which do not contain the fingerprint information.

```
<ibm>
  <Rfp>uuid1</Rfp>
  <Rfp>uuid2</Rfp>
  <Rfp>uuid3</Rfp>
  . . .
</ibm>
```

<ibm> is the name of the folder that holds the list of routing fingerprints containing the unique user identifier (uuid) of each queue manager that has been visited.

Every time that a message is published by a queue manager, it adds its uuid into the <ibm> folder using the <Rfp> (routing fingerprint) tag. Whenever a publication is received, IBM MQ uses the message properties API to iterate through the <Rfp> tags to see if that particular uuid value is present. Because of the way that the WebSphere Platform Messaging component of IBM MQ attaches to IBM Integration Bus through a channel and RFH2 subscription when using the queued publish/subscribe interface, IBM MQ also creates a fingerprint when it receives a publication by that route.

The goal is to not deliver any RFH2 to an application if it is not expecting any, simply because we have added in our fingerprint information.

Whenever an RFH2 is converted into message properties, it will also be necessary to convert the <ibm> folder; this removes the fingerprint information from the RFH2 that is passed on or delivered to applications that have used the IBM MQ V7.0, or later, API.

JMS applications do not see the fingerprint information, because the JMS interface does not extract that information from the RFH2, and therefore does not hand it on to its applications.

The Rfp message properties are created with `propDesc.CopyOptions = MQCOPY_FORWARD` and `MQCOPY_PUBLISH`. This has implications for applications receiving and then republishing the same message. It means that such an application can continue the chain of routing fingerprints by using `PutMsgOpts.Action = MQACTP_FORWARD`, but must be coded appropriately to remove its own fingerprint from the chain. By default the application uses `PutMsgOpts.Action = MQACTP_NEW` and starts a new chain.

IBM MQ client for HP Integrity NonStop Server troubleshooting

Provides information to help you to detect and deal with problems when you are using the IBM MQ client for HP Integrity NonStop Server.

Toggling between the use of IBM MQ and TMF transactions on a single connection

If an IBM MQ client for HP Integrity NonStop Server application toggles between the use of IBM MQ and TMF transactions on a single connection, then IBM MQ operations such as MQPUT and MQGET might fail with a return code of "2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE" on page 1352. Errors and a first failure symptom report for the client application are generated in the IBM MQ client for HP Integrity NonStop Server errors directory.

This error occurs because mixed TMF and IBM MQ transactions on a single connection are not supported.

Use the standard facilities that are supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM MQ Support site: http://www.ibm.com/software/integration/wmq/support/, or the IBM Support Assistant (ISA): http://www.ibm.com/software/support/isa/ to check whether a solution is already available. If you are unable to find a solution, contact your IBM support center. Do not discard these files until the problem is resolved.

Java and JMS troubleshooting

Use the advice that is given here to help you to resolve common problems that can arise when you are using Java or JMS applications.

About this task

The subtopics in this section provide advice to help you to detect and deal with problems that you might encounter under the following circumstances:
- When using the IBM MQ resource adapter
- When connecting to a queue manager with a specified provider version

Related concepts:

"Tracing IBM MQ classes for JMS applications" on page 1709
The trace facility in IBM MQ classes for JMS is provided to help IBM Support to diagnose customer issues. Various properties control the behavior of this facility.

"Tracing the IBM MQ Resource Adapter" on page 1717
The ResourceAdapter object encapsulates the global properties of the IBM MQ resource adapter. To enable trace of the IBM MQ resource adapter, properties need to be defined in the ResourceAdapter object.

"Tracing additional IBM MQ Java components" on page 1718
For Java components of IBM MQ, for example the IBM MQ Explorer and the Java implementation of IBM MQ Transport for SOAP, diagnostic information is output using the standard IBM MQ diagnostic facilities or by Java diagnostic classes.

Related tasks:

"Tracing IBM MQ classes for Java applications" on page 1713
The trace facility in IBM MQ classes for Java is provided to help IBM Support to diagnose customer issues. Various properties control the behavior of this facility.

Related information:

Using IBM MQ classes for JMS

Using the IBM MQ resource adapter

Using IBM MQ classes for Java

JMS provider version troubleshooting

Use the advice that is given here to help you to resolve common problems that can arise when you are connecting to a queue manager with a specified provider version.

JMS 2.0 function is not supported with this connection error

- **Error code:** JMSCC5008
- **Scenario:** You receive a `JMS 2.0 function is not supported with this connection` error.
- **Explanation:** The use of the JMS 2.0 functionality is only supported when connecting to an IBM MQ Version 8.0 queue manager that is using IBM MQ messaging provider Version 8 mode.
- **Solution:** Change the application to not use the JMS 2.0 function, or ensure that the application connects to an IBM MQ Version 8.0 queue manager that is using IBM MQ messaging provider Version 8 mode.

JMS 2.0 API is not supported with this connection error

- **Error code:** JMSCC5007
- **Scenario:** You receive a `JMS 2.0 API is not supported with this connection` error.
- **Explanation:** The use of the JMS 2.0 API is only supported when you are connecting to an IBM MQ Version 7 or Version 8 queue manager that is using IBM MQ messaging provider Normal or Version 8 mode. You might, for example, receive this error if you are attempting to connect to a Version 6 queue manager or if you are connecting by using migration mode. This typically happens if SHARECNV(0) or PROVIDER_VERSION=6 is specified.
- **Solution:** Change the application to not use the JMS 2.0 API, or ensure that the application connects to an IBM MQ Version 7 or Version 8 queue manager by using IBM MQ messaging provider Normal or Version 8 mode.

Queue manager command level did not match the requested provider version error

- **Error code:** JMSFMQ0003
- **Scenario:** You receive a `queue manager command level did not match the requested provider version`error.

- **Explanation:** The queue manager version that is specified in the provider version property on the connection factory is not compatible with the requested queue manager. For example, you might have specified PROVIDER_VERSION=8, and attempt to connect to a queue manager with a command level less than 800, such as 750.
- **Solution:** Modify the connection factory to connect to a queue manager that can support the provider version required.

For more information about provider version, see Rules for selecting the IBM MQ messaging provider mode.

Related information:

Configuring the JMS **PROVIDERVERSION** property

PCF processing in JMS

IBM MQ Programmable Change Format (PCF) messages are a flexible, powerful way in which to query and modify attributes of a queue manager, and the PCF classes provided in the IBM MQ classes for Java provide a convenient way of accessing their functionality in a Java application. The functionality can also be accessed from IBM MQ classes for JMS, however there is a potential problem.

The common model for processing PCF responses in JMS

A common approach to processing PCF responses in JMS is to extract the bytes payload of the message, wrap it in a `DataInputStream` and pass it to the `com.ibm.mq.headers.pcf.PCFMessage` constructor.

```
Message m = consumer.receive(10000);
//Reconstitute the PCF response.
ByteArrayInputStream bais =
    new ByteArrayInputStream(((BytesMessage)m).getBody(byte[].class));
DataInput di = new DataInputStream(bais);
 PCFMessage pcfResponseMessage = new PCFMessage(di);
```

See Using the IBM MQ Headers package for some examples.

Unfortunately this is not a totally reliable approach for all platforms - in general the approach works for big-endian platforms, but not for little-endian platforms.

What is the problem?

The problem is that in parsing the message headers, the `PCFMessage` class has to deal with issues of numeric encoding - the headers contain length fields which are in some encoding, that is big-endian or little-endian.

If you pass a "pure" `DataInputStream` to the constructor, the `PCFMessage` class has no good indication of the encoding, and has to assume a default - quite possibly incorrectly.

If this situation arises, you will probably see a "MQRCCF_STRUCTURE_TYPE_ERROR" (reason code 3013) in the constructor:

```
com.ibm.mq.headers.MQDataException: MQJE001: Completion Code '2', Reason '3013'.
        at com.ibm.mq.headers.pcf.PCFParameter.nextParameter(PCFParameter.java:167)
        at com.ibm.mq.headers.pcf.PCFMessage.initialize(PCFMessage.java:854)
        at com.ibm.mq.headers.pcf.PCFMessage.<init>(PCFMessage.java:156)
```

This message almost invariably means that the encoding has been misinterpreted. The probable reason for this is that the data that has been read is little-endian data which has been interpreted as big-endian.

The solution

The way to avoid this problem is to pass the `PCFMessage` constructor something which will tell the constructor the numeric encoding of the data it is working with.

To do this, make an MQMessage from the data received.

The following code is an outline example of the code you might use.

Attention: The code is an outline example only and does not contain any error handling information.

```
// get a response into a JMS Message
Message receivedMessage = consumer.receive(10000);
BytesMessage bytesMessage = (BytesMessage) receivedMessage;
byte[] bytesreceived = new byte[(int) bytesMessage.getBodyLength()];
bytesMessage.readBytes(bytesreceived);

// convert to MQMessage then to PCFMessage
MQMessage mqMsg = new MQMessage();
mqMsg.write(bytesreceived);
mqMsg.encoding = receivedMessage.getIntProperty("JMS_IBM_Encoding");
mqMsg.format = receivedMessage.getStringProperty("JMS_IBM_Format");
mqMsg.seek(0);

PCFMessage pcfMsg = new PCFMessage(mqMsg);
```

JMS connection pool error handling

Connection pool error handling is carried out by various methods of a purge policy.

The connection pool purge policy comes into operation if an error is detected when an application is using a JMS connection to a JMS provider. The connection manager can either:

- Close only the connection that encountered the problem. This is known as the FailingConnectionOnly purge policy and is the default behavior.

  Any other connections created from the factory, that is, those in use by other applications, and those that are in the free pool of the factory, are left alone.
- Close the connection that encountered the problem, throw away any connections in the free pool of the factory, and mark any in-use connections as stale.

  The next time the application that is using the connection tries to perform a connection-based operation, the application receives a StaleConnectionException. For this behavior, set the purge policy to Entire Pool.

Purge policy - failing connection only

Use the example described in How MDB listener ports use the connection pool. Two MDBs are deployed into the application server, each one using a different listener port. The listener ports both use the jms/CF1 connection factory.

After 600 seconds, you stop the first listener, and the connection that this listener port was using is returned to the connection pool.

If the second listener encounters a network error while polling the JMS destination, the listener port shuts down. Because the purge policy for the jms/CF1 connection factory is set to FailingConnectionOnly, the connection manager throws away only the connection that was used by the second listener. The connection in the free pool remains where it is.

If you now restart the second listener, the connection manager passes the connection from the free pool to the listener.

Purge policy - entire pool

For this situation, assume that you have three MDBs installed into your application server, each one using its own listener port. The listener ports have created connections from the jms/CF1 factory. After a period of time you stop the first listener, and its connection, c1, is put into the jms/CF1 free pool.

When the second listener detects a network error, it shuts itself down and closes `c2`. The connection manager now closes the connection in the free pool. However, the connection being used by third listener remains.

What should you set the purge policy to?

As previously stated, the default value of the purge policy for JMS connection pools is `FailingConnectionOnly`.

However, setting the purge policy to `EntirePool` is a better option. In most cases, if an application detects a network error on its connection to the JMS provider, it is likely that all open connections created from the same connection factory have the same problem.

If the purge policy is set to `FailingConnectionOnly`, the connection manager leaves all of the connections in the free pool. The next time an application tries to create a connection to the JMS provider, the connection manager returns one from the free pool if there is one available. However, when the application tries to use the connection, it encounters the same network problem as the first application.

Now, consider the same situation with the purge policy set to `EntirePool`. As soon as the first application encounters the network problem, the connection manager discards the failing connection and closes all connections in the free pool for that factory.

When a new application starts up and tries to create a connection from the factory, the connection manager tries to create a new one, as the free pool is empty. Assuming that the network problem has been resolved, the connection returned to the application is valid.

Troubleshooting JMSCC0108 messages

There are a number of steps that you can take to prevent a `JMSCC0108` message from occurring when you are using activation specifications and WebSphere Application Server listener ports that are running in Application Server Facilities (ASF) mode.

When you are using activation specifications and WebSphere Application Server listener ports that are running in ASF mode, which is the default mode of operation, it is possible that the following message might appear in the application server log file:

```
JMSCC0108: The IBM MQ classes for JMS had detected a message, ready for asynchronous delivery to an application.
When delivery was attempted, the message was no longer available.
```

Use the information in this topic to understand why this message appears, and the possible steps that you can take to prevent it from occurring.

How activation specifications and listener ports detect and process messages

An activation specification or WebSphere Application Server listener port performs the following steps when it starts up:

1. Create a connection to the queue manager that they have been set to use.
2. Open the JMS destination on that queue manager that they have been configured to monitor.
3. Browse that destination for messages.

When a message is detected, the activation specification or listener port performs the following steps:

1. Constructs an internal message reference that represents the message.
2. Gets a server session from its internal server session pool.
3. Loads the server session up with the message reference.
4. Schedules a piece of work with the application server Work Manager to run the server session and process the message.

The activation specification or listener port then goes back to monitoring the destination again, looking for another message to process.

The application server Work Manager runs the piece of work that the activation specification or listener port submitted on a new server session thread. When started, the thread completes the following actions:

- Starts either a local or global (XA) transaction, depending on whether the message-driven bean requires XA transactions or not, as specified in the message-driven bean's deployment descriptor.
- Gets the message from the destination by issuing a destructive MQGET API call.
- Runs the message-driven bean's onMessage() method.
- Completes the local or global transaction, once the onMessage() method has finished.
- Return the server session back to the server session pool.

Why the JMSCC0108 message occurs, and how to prevent it

The main activation specification or listener port thread browses messages on a destination. It then asks the Work Manager to start a new thread to destructively get the message and process it. This means that it is possible for a message to be found on a destination by the main activation specification or listener port thread, and no longer be available by the time the server session thread attempts to get it. If this happens, then the server session thread writes the following message to the application server's log file:

```
JMSCC0108: The IBM MQ classes for JMS had detected a message, ready for asynchronous delivery to an application.
When delivery was attempted, the message was no longer available.
```

There are two reasons why the message is no longer on the destination when the server session thread tries to get it:

- Reason 1: The message has been consumed by another application
- Reason 2: The message has expired

Reason 1: The message has been consumed by another application

If two or more activation specifications and/or listener ports are monitoring the same destination, then it is possible that they could detect the same message and try to process it. When this happens:

- A server session thread started by one activation specification or listener port gets the message and delivers it to a message-driven bean for processing.
- The sever session thread started by the other activation specification or listener port tries to get the message, and finds that it is no longer on the destination.

If an activation specification or listener port is connecting to a queue manager in any of the following ways, the messages that the main activation specification or listener port thread detects are marked:

- A queue manager on any platform, using IBM MQ messaging provider normal mode.
- A queue manager on any platform, using IBM MQ messaging provider normal mode with restrictions
- A queue manager running on z/OS, using IBM MQ messaging provider migration mode.

Marking a message prevents any other activation specification or listener port from seeing that message, and trying to process it.

By default, messages are marked for five seconds. After the message has been detected and marked, the five second timer starts. During these five seconds, the following steps must be carried out:

- The activation specification or listener port must get a server session from the server session pool.
- The server session must be loaded with details of the message to process.
- The work must be scheduled.
- The Work Manager must process the work request and start the server session thread.
- The server session thread needs to start either a local or global transaction.

- The server session thread needs to destructively get the message.

On a busy system, it might take longer than five seconds for these steps to be carried out. If this happens, then the mark on the message is released. This means that other activation specifications or listener ports can now see the message, and can potentially try to process it, which can result in the `JMSCC0108` message being written to the application server's log file.

In this situation, you should consider the following options:

- Increase the value of the queue manager property Message mark browse interval (MARKINT), to give the activation specification or listener port that originally detected the message more time to get it. Ideally, the property should be set to a value greater than the time taken for your message-driven beans to process messages. This means that, if the main activation specification or listener port thread blocks waiting for a server session because all of the server sessions are busy processing messages, then the message should still be marked when a server session becomes available. Note that the MARKINT property is set on a queue manager, and so is applicable to all applications that browse messages on that queue manager.
- Increase the size of the server session pool used by the activation specification or listener port. This would mean that there are more server sessions available to process messages, which should ensure that messages can be processed within the specified mark interval. One thing to note with this approach is that the activation specification or listener port will now be able to process more messages concurrently, which could impact the overall performance of the application server.

If an activation specification or listener port is connecting to a queue manager running on a platform other than z/OS, using IBM MQ messaging provider migration mode, the marking functionality is not available. This means that it is not possible to prevent two or more activation specifications and/or listener ports from detecting the same message and trying to process it. In this situation, the `JMSCC0108` message is expected.

Reason 2: The message has expired

The other reason that a `JMSCC0108` message is generated is if the message has expired in between being detected by the activation specification or listener port and being consumed by the server session. If this happens, when the server session thread tries to get the message, it finds that it is no longer there and so reports the `JMSCC0108` message.

Increasing the size of the server session pool used by the activation specification or listener port can help here. Increasing the server session pool size means that there are more server sessions available to process messages, which can potentially mean that the message is processed before it expires. It is important to note that the activation specification or listener port is now able to process more messages concurrently, which could impact the overall performance of the application server.

Problem determination for the IBM MQ resource adapter

When using the IBM MQ resource adapter, most errors cause exceptions to be thrown, and these exceptions are reported to the user in a manner that depends on the application server. The resource adapter makes extensive use of linked exceptions to report problems. Typically, the first exception in a chain is a high-level description of the error, and subsequent exceptions in the chain provide the more detailed information that is required to diagnose the problem.

For example, if the IVT program fails to obtain a connection to a IBM MQ queue manager, the following exception might be thrown:

```
javax.jms.JMSException: MQJCA0001: An exception occurred in the JMS layer.
See the linked exception for details.
```

Linked to this exception is a second exception:

```
javax.jms.JMSException: MQJMS2005: failed to create an MQQueueManager for
'localhost:ExampleQM'
```

This exception is thrown by IBM MQ classes for JMS and has a further linked exception:

```
com.ibm.mq.MQException: MQJE001: An MQException occurred: Completion Code 2,
Reason 2059
```

This final exception indicates the source of the problem. Reason code 2059 is MQRC_Q_MGR_NOT_AVAILABLE, which indicates that the queue manager specified in the definition of the ConnectionFactory object might not have been started.

If the information provided by exceptions is not sufficient to diagnose a problem, you might need to request a diagnostic trace. For information about how to enable diagnostic tracing, see Configuration of the IBM MQ resource adapter.

Configuration problems commonly occur in the following areas:
- Deploying the resource adapter
- Deploying MDBs
- Creating connections for outbound communication

Related information:

Using the IBM MQ resource adapter

Problems in deploying the resource adapter

If the resource adapter fails to deploy, check that Java EE Connector Architecture (JCA) resources are configured correctly. If IBM MQ is already installed, check that the correct versions of the JCA and IBM MQ classes for JMS are in the class path.

Failures in deploying the resource adapter are generally caused by not configuring JCA resources correctly. For example, a property of the ResourceAdapter object might not be specified correctly, or the deployment plan required by the application server might not be written correctly. Failures might also occur when the application server attempts to create objects from the definitions of JCA resources and bind the objects into the Java Naming Directory Interface (JNDI) namespace, but certain properties are not specified correctly or the format of a resource definition is incorrect.

The resource adapter can also fail to deploy because it loaded incorrect versions of JCA or IBM MQ classes for JMS classes from JAR files in the class path. This type of failure can commonly occur on a system where IBM MQ is already installed. On such a system, the application server might find existing copies of the IBM MQ classes for JMS JAR files and load classes from them in preference to the classes supplied in the IBM MQ resource adapter RAR file.

Related information:

What is installed for IBM MQ classes for JMS

Configuring the application server to use the latest resource adapter maintenance level

Problems in deploying MDBs

Failures when the application server attempts to start message delivery to an MDB might be caused by an error in the definition of the associated ActivationSpec object, or by missing resources.

Failures might occur when the application server attempts to start message delivery to an MDB. This type of failure is typically caused by an error in the definition of the associated ActivationSpec object, or because the resources referenced in the definition are not available. For example, the queue manager might not be running, or a specified queue might not exist.

An ActivationSpec object attempts to validate its properties when the MDB is deployed. Deployment then fails if the ActivationSpec object has any properties that are mutually exclusive or does not have all the required properties. However, not all problems associated with the properties of the ActivationSpec object can be detected at this time.

Failures to start message delivery are reported to the user in a manner that depends on the application server. Typically, these failures are reported in the logs and diagnostic trace of the application server. If enabled, the diagnostic trace of the IBM MQ resource adapter also records these failures.

Problems in creating connections for outbound communication

A failure in outbound communication can occur if a ConnectionFactory object cannot be found, or if the ConnectionFactory object is found but a connection cannot be created. There are various reasons for either of these problems.

Failures in outbound communication typically occur when an application attempts to look up and use a ConnectionFactory object in a JNDI namespace. A JNDI exception is thrown if the ConnectionFactory object cannot be found in the namespace. A ConnectionFactory object might not be found for the following reasons:

- The application specified an incorrect name for the ConnectionFactory object.
- The application server was not able to create the ConnectionFactory object and bind it into the namespace. In this case, the startup logs of the application server typically contain information about the failure.

If the application successfully retrieves the ConnectionFactory object from the JNDI namespace, an exception might still be thrown when the application calls the ConnectionFactory.createConnection() method. An exception in this context indicates that it is not possible to create a connection to an IBM MQ queue manager. Here are some common reasons why an exception might be thrown:

- The queue manager is not available, or cannot be found using the properties of the ConnectionFactory object. For example, the queue manager is not running, or the specified host name, IP address, or port number of the queue manager is incorrect.
- The user is not authorized to connect to the queue manager. For a client connection, if the createConnection() call does not specify a user name, and the application server supplies no user identity information, the JVM process ID is passed to the queue manager as the user name. For the connection to succeed, this process ID must be a valid user name in the system on which the queue manager is running.
- The ConnectionFactory object has a property called ccdtURL and a property called channel. These properties are mutually exclusive.
- On an SSL connection, the SSL-related properties, or the SSL-related attributes in the server connection channel definition, have not been specified correctly.
- The sslFipsRequired property has different values for different JCA resources. For more information about this limitation, see Limitations of the IBM MQ resource adapter.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

Using IBM MQ connection property override

V 8.0.0.4

Connection property override allows you to change the details that are used by a client application to connect to a queue manager, without modifying the source code.

About this task

Sometimes, it is not possible to modify the source code for an application, for example, if the application is a legacy application and the source code is no longer available.

In this situation, if an application needs to specify different properties when it is connecting to a queue manager, or is required to connect to a different queue manager, then you can use the connection override functionality to specify the new connection details or queue manager name.

The connection property override is supported for two clients:
- IBM MQ classes for JMS
- IBM MQ classes for Java

You can override the properties that you want to change by defining them in a configuration file that is then read by the IBM MQ classes for JMS or IBM MQ classes for Java at startup.

When the connection override functionality is in use, all applications that are running inside the same Java runtime environment pick up and use the new property values. If multiple applications that are using either the IBM MQ classes for JMS or the IBM MQ classes for Java are running inside the same Java runtime environment, it is not possible to just override properties for individual applications.

Important: This functionality is only supported for situations where it is not possible to modify the source code for an application. It must not be used for applications where the source code is available and can be updated.

Related concepts:

"Tracing IBM MQ classes for JMS applications" on page 1709
The trace facility in IBM MQ classes for JMS is provided to help IBM Support to diagnose customer issues. Various properties control the behavior of this facility.

Related tasks:

"Tracing IBM MQ classes for Java applications" on page 1713
The trace facility in IBM MQ classes for Java is provided to help IBM Support to diagnose customer issues. Various properties control the behavior of this facility.

Related information:

Using IBM MQ classes for JMS

Using IBM MQ classes for Java

Using connection property override in IBM MQ classes for JMS

V 8.0.0.4

If a connection factory is created programmatically, and it is not possible to modify the source code for the application that creates it, then the connection override functionality can be used to change the properties that the connection factory uses when a connection is created. However, the use of the connection override functionality with connection factories defined in JNDI is not supported.

About this task

In the IBM MQ classes for JMS, details about how to connect to a queue manager are stored in a connection factory. Connection factories can either be defined administratively and stored in a JNDI repository, or created programmatically by an application by using Java API calls.

If an application creates a connection factory programmatically, and it is not possible to modify the source code for that application, the connection override functionality allows you to override the connection factory properties in the short term. In the long term, though, you must put plans in place to allow the connection factory used by the application to be modified without using the connection override functionality.

If the connection factory that is created programmatically by an application is defined to use a Client Channel Definition Table (CCDT), then the information in the CCDT is used in preference to the overridden properties. If the connection details that the application uses need to be changed, then a new version of the CCDT must be created and made available to the application.

The use of the connection override functionality with connection factories defined in JNDI is not supported. If an application uses a connection factory that is defined in JNDI, and the properties of that connection factory need to be changed, then the definition of the connection factory must be updated in JNDI. Although the connection override functionality is applied to these connection factories (and the overridden properties take precedence over the properties in the connection factory definition that is looked up in JNDI), this use of the connection override functionality is not supported.

Important: The connection override functionality affects all of the applications that are running inside of a Java runtime environment, and applies to all of the connection factories used by those applications. It is not possible to just override properties for individual connection factories or applications.

When an application uses a connection factory to create a connection to a queue manager, the IBM MQ classes for JMS look at the properties that have been overridden and use those property values when creating the connection, rather than the values for the same properties in the connection factory.

For example, suppose a connection factory has been defined with the PORT property set to 1414. If the connection override functionality has been used to set the PORT property to 1420, then when the connection factory is used to create a connection, the IBM MQ classes for JMS use a value of 1420 for the PORT property, rather than 1414.

To modify any of the connection properties that are used when creating a JMS connection from a connection factory, the following steps need to be carried out:
1. Add the properties to be overridden to an IBM MQ classes for JMS configuration file.
2. Enable the connection override functionality.
3. Start the application, specifying the configuration file.

Procedure

1. Add the properties to be overridden to an IBM MQ classes for JMS configuration file.
   a. Create a file containing the properties and values that need to be overridden in the standard Java properties format. For details about how you create a properties file, see The IBM MQ classes for JMS configuration file.
   b. To override a property, add an entry to the properties file. Any IBM MQ classes for JMS connection factory property can be overridden. Add each required entry in the following format:
   ```
   jmscf.<property name>=<value>
   ```

where *<property name>* is the JMS administration property name or XMSC constant for the property that needs to be overridden. For a list of connection factory properties, see Properties of IBM MQ classes for JMS objects.

For example, to set the name of the channel that an application should use to connect to a queue manager, you can add the following entry to the properties file:

```
jmscf.channel=MY.NEW.SVRCONN
```

2. Enable the connection override functionality. To enable connection override, set the **com.ibm.msg.client.jms.overrideConnectionFactory** property to be true so that the properties that are specified in the properties file are used to override the values that are specified in the application. You can either set the extra property as another property in the configuration file itself, or pass the property as a Java system property by using:

```
-Dcom.ibm.msg.client.jms.overrideConnectionFactory=true
```

3. Start the application, specifying the configuration file. Pass the properties file that you created to the application at run time by setting the Java system property:

```
-Dcom.ibm.msg.client.config.location
```

Note that the location of the configuration file must be specified as a URI, for example:

```
-Dcom.ibm.msg.client.config.location=file:///jms/jms.config
```

Results

When the connection override functionality is enabled, the IBM MQ classes for JMS write an entry to the jms log whenever a connection is made. The information in the log shows the connection factory properties that were overridden when the connection was created, as shown in the following example entry:

```
Overriding ConnectionFactory properties:
        Overriding property channel:
             Original value = MY.OLD.SVRCONN
             New value      = MY.NEW.SVRCONN
```

Related tasks:

"Using connection property override in IBM MQ classes for Java"
In the IBM MQ classes for Java, connection details are set as properties using a combination of different values. The connection override functionality can be used to override the connection details that an application uses if it is not possible to modify the source code for the application.

"Overriding connection properties: example with IBM MQ classes for JMS" on page 1799
This example shows how to override properties when you are using the IBM MQ classes for JMS.

Related information:

Creating and configuring connection factories and destinations in an IBM MQ classes for JMS application

Configuring connection factories and destinations in a JNDI namespace

Using connection property override in IBM MQ classes for Java

V 8.0.0.4

In the IBM MQ classes for Java, connection details are set as properties using a combination of different values. The connection override functionality can be used to override the connection details that an application uses if it is not possible to modify the source code for the application.

About this task

The different values that are used to set the connection properties are a combination of:

- Assigning values to static fields on the **MQEnvironment** class.

- Setting property values in the properties `Hashtable` in the **`MQEnvironment`** class.
- Setting property values in a `Hashtable` passed into an **`MQQueueManager`** constructor.

These properties are then used when an application constructs an MQQueueManager object, which represents a connection to a queue manager.

If it is not possible to modify the source code for an application that uses the IBM MQ classes for Java to specify different properties that must be used when creating a connection to a queue manager, the connection override functionality allows you to override the connection details in the short term. In the long term, though, you must put plans in place to allow the connection details used by the application to be modified without using the connection override functionality.

When an application creates an MQQueueManager, the IBM MQ classes for Java look at the properties that have been overridden and use those property values when creating a connection to the queue manager, rather than the values in any of the following locations:
- The static fields on the MQEnvironment class
- The properties Hashtable stored in the MQEnvironment class
- The properties Hashtable that is passed into an MQQueueManager constructor

For example, suppose an application creates an MQQueueManager, passing in a properties Hashtable that has the CHANNEL property set to MY.OLD.CHANNEL. If the connection override functionality has been used to set the CHANNEL property to MY.NEW.CHANNEL, then when the MQQueueManager is constructed, the IBM MQ classes for Java attempt to create a connection to the queue manager by using the channel MY.NEW.CHANNEL rather than MY.OLD.CHANNEL.

Note: If an MQQueueManager is configured to use a Client Channel Definition Table (CCDT), then the information in the CCDT is used in preference to the overridden properties. If the connection details that the application creating the MQQueueManager uses need to be changed, then a new version of the CCDT must be created and made available to the application.

To modify any of the connection properties that are used when creating an MQQueueManager, the following steps need to be carried out:
1. Create a properties file called `mqclassesforjava.config`.
2. Enable the connection property override functionality by setting the **`OverrideConnectionDetails`** property to true.
3. Start the application, specifying the configuration file as part of the Java invocation.

Procedure

1. Create a properties file called `mqclassesforjava.config` containing the properties and values that need to be overridden. It is possible to override 13 properties that are used by the IBM MQ classes for Java when connecting to a queue manager as part of the MQQueueManager constructor. The names of these properties, and the keys that must be specified when you are overriding them, are shown in the following table:

Table 173. Properties that can be overridden

Property	Property key
CCSID	$CCSID_PROPERTY
Channel	$CHANNEL_PROPERTY
Connect options	$CONNECT_OPTIONS_PROPERTY
Hostname	$HOST_NAME_PROPERTY
SSL key reset	$SSL_RESET_COUNT_PROPERTY
Local address	$LOCAL_ADDRESS_PROPERTY
Queue manager name	qmgr
Password	$PASSWORD_PROPERTY
Port	$PORT_PROPERTY
Cipher suite	$SSL_CIPHER_SUITE_PROPERTY
FIPS required	$SSL_FIPS_REQUIRED_PROPERTY
SSL peer name	$SSL_PEER_NAME_PROPERTY
User ID	$USER_ID_PROPERTY

Note: All of the property keys start with the $ character, except for the queue manager name. The reason for this is because the queue manager name is passed in to the MQQueueManager constructor as an argument, rather than being set as either a static field on the MQEnvironment class, or a property in a Hashtable, and so internally this property needs to be treated slightly differently from the other properties.

To override a property, add an entry in the following format to the properties file:

```
mqj.<property key>=<value>
```

For example, to set the name of the channel to be used when creating MQQueueManager objects, you can add the following entry to the properties file:

```
mqj.$CHANNEL_PROPERTY=MY.NEW.CHANNEL
```

To change the name of the queue manager that an MQQueueManager object connects to, you can add the following entry to the properties file:

```
mqj.qmgr=MY.OTHER.QMGR
```

2. Enable the connection override functionality by setting the **`com.ibm.mq.overrideConnectionDetails`** property to be true. Setting the property **`com.ibm.mq.overrideConnectionDetails`** to be true means that the properties that are specified in the properties file are used to override the values specified in the application. You can either set the extra property as another property in the configuration file itself, or pass the property as a system property, by using:

```
-Dcom.ibm.mq.overrideConnectionDetails=true
```

3. Start the application. Pass the properties file you created to the client application at run time by setting the Java system property:

```
-Dcom.ibm.msg.client.config.location
```

Note that the location of the configuration file must be specified as a URI, for example:

```
-Dcom.ibm.msg.client.config.location=file:///classesforjava/mqclassesforjava.config
```

Overriding connection properties: example with IBM MQ classes for JMS

V 8.0.0.4

This example shows how to override properties when you are using the IBM MQ classes for JMS.

About this task

The following code example shows how an application creates a ConnectionFactory programmatically:

```
JmsSampleApp.java
...
JmsFactoryFactory jmsff;
JmsConnectionFactory jmsConnFact;

jmsff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
jmsConnFact = jmsff.createConnectionFactory();

jmsConnFact.setStringProperty(WMQConstants.WMQ_HOST_NAME,"127.0.0.1");
jmsConnFact.setIntProperty(WMQConstants.WMQ_PORT, 1414);
jmsConnFact.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER,"QM_V80");
jmsConnFact.setStringProperty(WMQConstants.WMQ_CHANNEL,"MY.CHANNEL");
jmsConnFact.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
                           WMQConstants.WMQ_CM_CLIENT);
...
```

The ConnectionFactory is configured to connect to the queue manager QM_V80 using the CLIENT transport and channel MY.CHANNEL.

You can override the connection details by using a properties file, and force the application to connect to a different channel, by using the following procedure.

Procedure

1. Create an IBM MQ classes for JMS configuration file that is called `jms.config` in the /*<userHome>* directory (where *<userHome>* is your home directory). Create this file with the following contents:

```
jmscf.CHANNEL=MY.TLS.CHANNEL
jmscf.SSLCIPHERSUITE=TLS_RSA_WITH_AES_128_CBC_SHA256
```

2. Run the application, passing the following Java system properties into the Java runtime environment that the application is running in:

```
-Dcom.ibm.msg.client.config.location=file:///<userHome>/jms.config
-Dcom.ibm.msg.client.jms.overrideConnectionFactory=true
```

Results

Carrying out this procedure overrides the ConnectionFactory that was created programmatically by the application, so that when the application creates a connection, it tries to connect by using the channel MY.TLS.CHANNEL and the cipher suite TLS_RSA_WITH_AES_128_CBC_SHA256.

Related tasks:

"Using IBM MQ connection property override" on page 1794
Connection property override allows you to change the details that are used by a client application to connect to a queue manager, without modifying the source code.

"Using connection property override in IBM MQ classes for JMS" on page 1794
If a connection factory is created programmatically, and it is not possible to modify the source code for the application that creates it, then the connection override functionality can be used to change the properties that the connection factory uses when a connection is created. However, the use of the connection override functionality with connection factories defined in JNDI is not supported.

"Using connection property override in IBM MQ classes for Java" on page 1796
In the IBM MQ classes for Java, connection details are set as properties using a combination of different values. The connection override functionality can be used to override the connection details that an application uses if it is not possible to modify the source code for the application.

Resolving problems with IBM MQ MQI clients

This collection of topics contains information about techniques for solving problems in IBM MQ MQI client applications.

An application running in the IBM MQ MQI client environment receives MQRC_* reason codes in the same way as IBM MQ server applications. However, there are additional reason codes for error conditions associated with IBM MQ MQI clients. For example:

- Remote machine not responding
- Communications line error
- Invalid machine address

The most common time for errors to occur is when an application issues an MQCONN or MQCONNX and receives the response MQRC_Q_MQR_NOT_AVAILABLE. Look in the client error log for a message explaining the failure. There might also be errors logged at the server, depending on the nature of the failure. Also, check that the application on the IBM MQ MQI client is linked with the correct library file.

IBM MQ MQI client fails to make a connection

An MQCONN or MQCONNX might fail because there is no listener program running on the server, or during protocol checking.

When the IBM MQ MQI client issues an MQCONN or MQCONNX call to a server, socket and port information is exchanged between the IBM MQ MQI client and the server. For any exchange of information to take place, there must be a program on the server with the role to 'listen' on the communications line for any activity. If there is no program doing this, or there is one but it is not configured correctly, the MQCONN or MQCONNX call fails, and the relevant reason code is returned to the IBM MQ MQI client application.

If the connection is successful, IBM MQ protocol messages are exchanged and further checking takes place. During the IBM MQ protocol checking phase, some aspects are negotiated while others cause the connection to fail. It is not until all these checks are successful that the MQCONN or MQCONNX call succeeds.

For information about the MQRC_* reason codes, see API reason codes.

Stopping IBM MQ MQI clients

Even though an IBM MQ MQI client has stopped, it is still possible for the associated process at the server to be holding its queues open. The queues are not closed until the communications layer detects that the partner has gone.

If sharing conversations is enabled, the server channel is always in the correct state for the communications layer to detect that the partner has gone.

Error messages with IBM MQ MQI clients

When an error occurs with an IBM MQ MQI client system, error messages are put into the IBM MQ system error files.

- On UNIX and Linux systems, these files are found in the `/var/mqm/errors` directory
- On Windows, these files are found in the errors subdirectory of the IBM MQ MQI client installation. Usually this directory is `C:\Program Files\IBM\WebSphere MQ\errors`.
- On IBM i, these files are found in the `/QIBM/UserData/mqm/errors` directory

Certain client errors can also be recorded in the IBM MQ error files associated with the server to which the client was connected.

Multicast troubleshooting

The following hints and tips are in no significant order, and might be added to when new versions of the documentation are released. They are subjects that, if relevant to the work that you are doing, might save you time.

Testing multicast applications on a non-multicast network

Use this information to learn how to test IBM MQ Multicast applications locally instead of over a multicast network.

When developing or testing multicast applications you might not yet have a multicast enabled network. To run the application locally, you must edit the `mqclient.ini` file as shown in the following example:

Edit the `Interface` parameter in the `Multicast` stanza of the *`MQ_DATA_PATH`* `/mqclient.ini`:

```
Multicast:
Interface      = 127.0.0.1
```

where *`MQ_DATA_PATH`* is the location of the IBM MQ data directory (`/var/mqm/mqclient.ini`).

The multicast transmissions now only use the local loopback adapter.

Setting the appropriate network for multicast traffic

When developing or testing multicast applications, after testing them locally, you might want to test them over a multicast enabled network. If the application only transmits locally, you might have to edit the `MQClient.ini` file as shown later in this section. If the machine setup is using multiple network adapters, or a virtual private network (VPN) for example, the **`Interface`** parameter in the `MQClient.ini` file must be set to the address of the network adapter you want to use.

If the `Multicast` stanza exists in the `MQClient.ini` file, edit the **`Interface`** parameter as shown in the following example:

Change:

```
Multicast:
Interface       = 127.0.0.1
```

To:

```
Multicast:
Interface       = IPAddress
```

where *IPAddress* is the IP address of the interface on which multicast traffic flows.

If there is no `Multicast` stanza in the `MQClient.ini` file, add the following example:

```
Multicast:
Interface       = IPAddress
```

where *IPAddress* is the IP address of the interface on which multicast traffic flows.

The multicast applications now run over the multicast network.

Multicast topic string is too long

If your IBM MQ Multicast topic string is rejected with reason code MQRC_TOPIC_STRING_ERROR, it might be because the string is too long.

WebSphereMQ Multicast has a 255 character limit for topic strings. This limitation means that care must be taken with the names of nodes and leaf-nodes within the tree; if the names of nodes and leaf-nodes are too long, the topic string might exceed 255 characters and return the “2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR” on page 1482 reason code. It is recommended to make topic strings as short as possible because longer topic strings might have a detrimental effect on performance.

Multicast topic topology issues

Use these examples to understand why certain IBM MQ Multicast topic topologies are not recommended.

As was mentioned in IBM MQ Multicast topic topology, IBM MQ Multicast support requires that each subtree has its own multicast group and data stream within the total hierarchy. Do not use a different multicast group address for a subtree and its parent.

The *classful network* IP addressing scheme has designated address space for multicast address. The full multicast range of IP address is `224.0.0.0` to `239.255.255.255`, but some of these addresses are reserved. For a list of reserved address either contact your system administrator or see http://www.iana.org/assignments/multicast-addresses for more information. It is recommended that you use the locally scoped multicast address in the range of `239.0.0.0` to `239.255.255.255`.

Recommended multicast topic topology

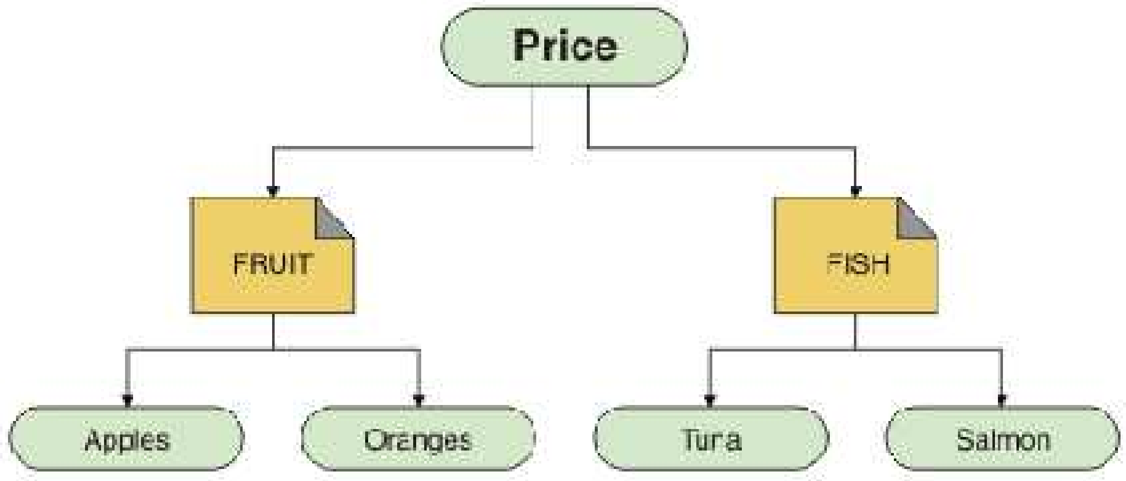
This example is the same as the one from IBM MQ Multicast topic topology, and shows 2 possible multicast data streams. Although it is a simple representation, it demonstrates the kind of situation that IBM MQ Multicast was designed for, and is shown here to contrast the second example:

```
DEF COMMINFO(MC1) GRPADDR(227.20.133.1)

DEF COMMINFO(MC2) GRPADDR(227.20.133.2)
```

where *227.20.133.1* and *227.20.133.2* are valid multicast addresses.

These topic definitions are used to create a topic tree as shown in the following diagram:

```
DEFINE TOPIC(FRUIT) TOPICSTRING('Price/FRUIT') MCAST(ENABLED) COMMINFO(MC1)

DEFINE TOPIC(FISH) TOPICSTRING('Price/FISH') MCAST(ENABLED) COMMINFO(MC2)
```



Each multicast communication information (COMMINFO) object represents a different stream of data because their group addresses are different. In this example, the topic `FRUIT` is defined to use COMMINFO object `MC1`, and the topic `FISH` is defined to use COMMINFO object `MC2` .

IBM MQ Multicast has a 255 character limit for topic strings. This limitation means that care must be taken with the names of nodes and leaf-nodes within the tree; if the names of nodes and leaf-nodes are too long, the topic string might exceed 255 characters and return the `MQRC_TOPIC_STRING_ERROR` reason code.

Non-recommended multicast topic topology

This example extends the previous example by adding another topic object called `ORANGES` which is defined to use another COMMINFO object definition (`MC3`):

```
DEF COMMINFO(MC1) GRPADDR(227.20.133.1
)

DEF COMMINFO(MC2) GRPADDR(227.20.133.2)

DEF COMMINFO(MC3) GRPADDR(227.20.133.3)
```

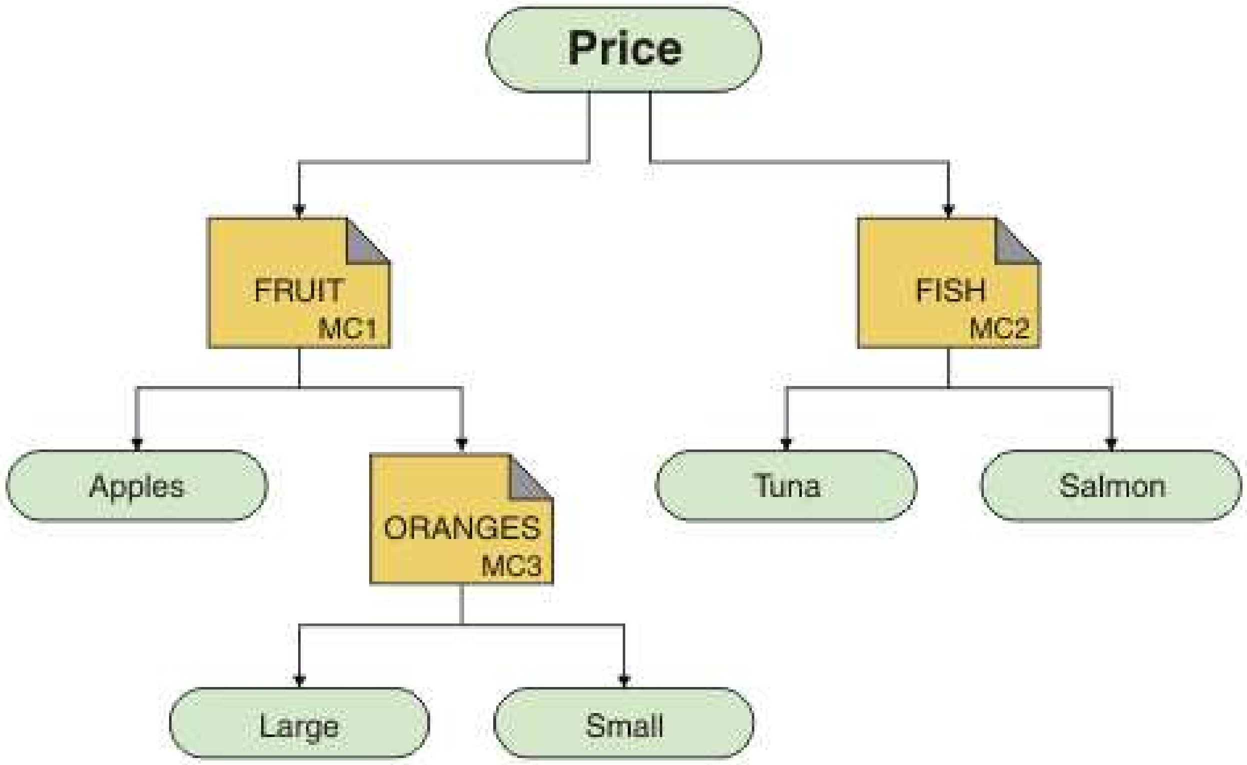
where *227.20.133.1*, *227.20.133.2*, and *227.20.133.3* are valid multicast addresses.

These topic definitions are used to create a topic tree as shown in the following diagram:

```
DEFINE TOPIC(FRUIT) TOPICSTRING('Price/FRUIT') MCAST(ENABLED) COMMINFO(MC1)
```

```
DEFINE TOPIC(FISH) TOPICSTRING('Price/FISH') MCAST(ENABLED) COMMINFO(MC2)
```

```
DEFINE TOPIC(ORANGES) TOPICSTRING('Price/FRUIT/ORANGES') MCAST(ENABLED) COMMINFO(MC3)
```



While this kind of multicast topology is possible to create, it is not recommended because applications might not receive the data that they were expecting.

An application subscribing on `'Price/FRUIT/#'` receives multicast transmission on the `COMMINFO MC1` group address. The application expects to receive publications on all topics at or below that point in the topic tree.

However, the messages created by an application publishing on `'Price/FRUIT/ORANGES/Small'` are not received by the subscriber because the messages are sent on the group address of `COMMINFO MC3`.

Queue manager clusters troubleshooting

Use the checklist given here, and the advice given in the subtopics, to help you to detect and deal with problems when you use queue manager clusters.

Before you begin

If your problems relate to publish/subscribe messaging using clusters, rather than to clustering in general, see "Routing for publish/subscribe clusters: Notes on behavior" on page 1780.

Procedure

- Check that your cluster channels are all paired.

  Each cluster sender channel connects to a cluster receiver channel of the same name. If there is no local cluster receiver channel with the same name as the cluster sender channel on the remote queue manager, then it won't work.
- Check that your channels are running. No channels should be in `RETRYING` state permanently.

  Show which channels are running using the following command:

  ```
  runmqsc display chstatus(*)
  ```

  If you have channels in `RETRYING` state, there might be an error in the channel definition, or the remote queue manager might not be running. While channels are in this state, messages are likely to build up on transmit queues. If channels to full repositories are in this state, then the definitions of cluster objects (for example queues and queue managers) become out-of-date and inconsistent across the cluster.
- Check that no channels are in `STOPPED` state.

  Channels go into `STOPPED` state when you stop them manually. Channels that are stopped can be restarted using the following command:

  ```
  runmqsc start channel(xyz)
  ```

  A clustered queue manager auto-defines cluster channels to other queue managers in a cluster, as required. These auto-defined cluster channels start automatically as needed by the queue manager, unless they were previously stopped manually. If an auto-defined cluster channel is stopped manually , the queue manager remembers that it was manually stopped and does not start it automatically in the future. If you need to stop a channel, either remember to restart it again at a convenient time, or else issue the following command:

  ```
  stop channel(xyz) status(inactive)
  ```

  The `status(inactive)` option allows the queue manager to restart the channel at a later date if it needs to do so.
- Check that all queue managers in the cluster are aware of all the full repositories.

  You can do this using the following command:

  ```
  runmqsc display clusqmgr(*) qmtype
  ```

  Partial repositories might not be aware of all other partial repositories. All full repositories should be aware of all queue managers in the cluster. If cluster queue managers are missing, this might mean that certain channels are not running correctly.
- Check that every queue manager (full repositories and partial repositories) in the cluster has a manually defined cluster receiver channel running and is defined in the correct cluster.

  To see which other queue managers are talking to a cluster receiver channel, use the following command:

  ```
  runmqsc display channel(*) rqmname
  ```

Check that each manually defined cluster receiver has a **conname** parameter defined to be `ipaddress(port)`. Without a correct connection name, the other queue manager does not know the connection details to use when connecting back.

- Check that every partial repository has a manually defined cluster sender channel running to a full repository, and defined in the correct cluster.

  The cluster sender channel name must match the cluster receiver channel name on the other queue manager.
- Check that every full repository has a manually defined cluster sender channel running to every other full repository, and defined in the correct cluster.

  The cluster sender channel name must match the cluster receiver channel name on the other queue manager. Each full repository does not keep a record of what other full repositories are in the cluster. It assumes that any queue manager to which it has a manually defined cluster sender channel is a full repository.
- Check the dead letter queue.

  Messages that the queue manager cannot deliver are sent to the dead letter queue.
- Check that, for each partial repository queue manager, you have defined a single cluster-sender channel to one of the full repository queue managers. This channel acts as a "bootstrap" channel through which the partial repository queue manager initially joins the cluster.
- Check that the intended full repository queue managers are actual full repositories and are in the correct cluster.

  You can do this using the following command:

  ```
  runmqsc display qmgr repos reposnl
  ```
- Check that messages are not building up on transmit queues or system queues.

  You can check transmit queues using the following command:

  ```
  runmqsc display ql(*) curdepth where (usage eq xmitq)
  ```

  You can check system queues using the following command:

  ```
  display ql(system*) curdepth
  ```

Related concepts:

"Making initial checks on Windows, UNIX and Linux systems" on page 1277
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

z/OS "Making initial checks on z/OS" on page 1297
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

IBM i "Making initial checks on IBM i" on page 1288
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

"Reason codes and exceptions" on page 1314
You can use the following messages and reason codes to help you solve problems with your IBM MQ components or applications.

Related information:

Configuring a queue manager cluster

A cluster-sender channel is continually trying to start

Check the queue manager and listener are running, and the cluster-sender and cluster-receiver channel definitions are correct.

Symptom

```
1 : display chs(*)
AMQ8417: Display Channel Status details.
CHANNEL(DEMO.QM2)                       XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(computer.ibm.com(1414))
CURRENT                                 CHLTYPE(CLUSSDR)
STATUS(RETRYING)
```

Cause

1. The remote queue manager is not available.
2. An incorrect parameter is defined either for the local manual cluster-sender channel or the remote cluster-receiver channel.

Solution

Check whether the problem is the availability of the remote queue manager.

1. Are there any error messages?
2. Is the queue manager active?
3. Is the listener running?
4. Is the cluster-sender channel able to start?

If the remote queue manager is available, is there a problem with a channel definition? Check the definition type of the cluster queue manager to see if the channel is continually trying to start; for example:

```
1 : dis clusqmgr(*) deftype where(channel eq DEMO.QM2)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM2) CHANNEL(DEMO.QM2) CLUSTER(DEMO)
DEFTYPE(CLUSSDRA)
```

If the definition type is `CLUSSDR` the channel is using the local manual cluster-sender definition. Alter any incorrect parameters in the local manual cluster-sender definition and restart the channel.

If the definition type is either `CLUSSDRA` or `CLUSSDRB` the channel is using an auto-defined cluster-sender channel. The auto-defined cluster-sender channel is based on the definition of a remote cluster receiver channel. Alter any incorrect parameters in the remote cluster receiver definition. For example, the `conname` parameter might be incorrect:

```
1 : alter chl(demo.qm2) chltype(clusrcvr) conname('newhost(1414)')
AMQ8016: WebSphere MQ channel changed.
```

Changes to the remote cluster-receiver definition are propagated out to any cluster queue managers that are interested. The corresponding auto-defined channels are updated accordingly. You can check that the updates have been propagated correctly by checking the changed parameter. For example:

```
1 : dis clusqmgr(qm2) conname
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM2) CHANNEL(DEMO.QM2) CLUSTER(DEMO) CONNAME(newhost(1414))
```

If the auto-defined definition is now correct, restart the channel.

DISPLAY CLUSQMGR shows CLUSQMGR names starting SYSTEM.TEMP.

The queue manager has not received any information from the full repository queue manager that the manually defined CLUSSDR channel points to. Check that the cluster channels are defined correctly.

Symptom

```
1 : display clusqmgr(*)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM1)             CLUSTER(DEMO)
CHANNEL(DEMO.QM1)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(SYSTEM.TEMPUUID.computer.hursley.ibm.com(1414))
CLUSTER(DEMO)             CHANNEL(DEMO.QM2)
```

Cause

The queue manager has not received any information from the full repository queue manager that the manually defined CLUSSDR channel points to. The manually defined `CLUSSDR` channel must be in running state.

Solution

Check that the `CLUSRCVR` definition is also correct, especially its `CONNAME` and `CLUSTER` parameters. Alter the channel definition, if the definition is wrong.

You also need to give the correct authority to the SYSTEM.CLUSTER.TRANSMIT.QUEUE by issuing the following command:

```
setmqaut -m <QMGR Name> -n SYSTEM.CLUSTER.TRANSMIT.QUEUE -t q -g mqm +all
```

It might take some time for the remote queue managers to attempt a new restart, and start their channels with the corrected definition.

Return code= 2035 `MQRC_NOT_AUTHORIZED`

The RC2035 reason code is displayed for various reasons including an error on opening a queue or a channel, an error received when you attempt to use a user ID that has administrator authority, an error when using an IBM MQ JMS application, and opening a queue on a cluster. MQS_REPORT_NOAUTH and MQSAUTHERRORS can be used to further diagnose RC2035.

Specific problems

See "Specific problems generating RC2035" on page 1335 for information on:
- JMSWMQ2013 invalid security authentication
- MQRC_NOT_AUTHORIZED on a queue or channel
- MQRC_NOT_AUTHORIZED (AMQ4036 on a client) as an administrator
- MQS_REPORT_NOAUTH and MQSAUTHERRORS environment variables

Opening a queue in a cluster

The solution for this error depends on whether the queue is on z/OS or not. On z/OS use your security manager. On other platforms create a local alias to the cluster queue, or authorize all users to have access to the transmission queue.

Symptom

Applications receive a return code of 2035 MQRC_NOT_AUTHORIZED when trying to open a queue in a cluster.

Cause

Your application receives the return code of MQRC_NOT_AUTHORIZED when trying to open a queue in a cluster. The authorization for that queue is correct. It is likely that the application is not authorized to put to the cluster transmission queue.

Solution

The solution depends on whether the queue is on z/OS or not. See the related information topic.

Return code= 2085 MQRC_UNKNOWN_OBJECT_NAME when trying to open a queue in the cluster

Symptom

Applications receive a return code of 2085 MQRC_UNKNOWN_OBJECT_NAME when trying to open a queue in the cluster.

Cause

The queue manager where the object exists or this queue manager might not have successfully entered the cluster.

Solution

Make sure that they can each display all the full repositories in the cluster. Also make sure that the CLUSSDR channels to the full repositories are trying to start.

If the queue is in the cluster, check that you have used appropriate open options. You cannot get messages from a remote cluster queue, so make sure that the open options are for output only.

```
1 : display clusqmgr(*) qmtype status
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM1)         CLUSTER(DEMO)
CHANNEL(DEMO.QM1)     QMTYPE(NORMAL)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM2)         CLUSTER(DEMO)
CHANNEL(DEMO.QM2)     QMTYPE(REPOS)
STATUS(RUNNING)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM3)         CLUSTER(DEMO)
CHANNEL(DEMO.QM3)     QMTYPE(REPOS)
STATUS(RUNNING)
```

Return code= 2189 `MQRC_CLUSTER_RESOLUTION_ERROR` when trying to open a queue in the cluster

Make sure that the CLUSSDR channels to the full repositories are not continually trying to start.

Symptom

Applications receive a return code of 2189 MQRC_CLUSTER_RESOLUTION_ERROR when trying to open a queue in the cluster.

Cause

The queue is being opened for the first time and the queue manager cannot contact any full repositories.

Solution

Make sure that the CLUSSDR channels to the full repositories are not continually trying to start.

```
1 : display clusqmgr(*) qmtype status
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM1)         CLUSTER(DEMO)
CHANNEL(DEMO.QM1)     QMTYPE(NORMAL)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM2)         CLUSTER(DEMO)
CHANNEL(DEMO.QM2)     QMTYPE(REPOS)
STATUS(RUNNING)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM3)         CLUSTER(DEMO)
CHANNEL(DEMO.QM3)     QMTYPE(REPOS)
STATUS(RUNNING)
```

Return code=2082 MQRC_UNKNOWN_ALIAS_BASE_Q opening a queue in the cluster

Applications get rc=2082 MQRC_UNKNOWN_ALIAS_BASE_Q when trying to open a queue in the cluster.

Problem

An MQOPEN or MQPUT1 call was issued specifying an alias queue as the target, but the *BaseQName* in the alias queue attributes is not recognized as a queue name.

This reason code can also occur when *BaseQName* is the name of a cluster queue that cannot be resolved successfully.

MQRC_UNKNOWN_ALIAS_BASE_Q might indicate that the application is specifying the **ObjectQmgrName** of the queue manager that it is connecting to, and the queue manager that is hosting the alias queue. This means that the queue manager looks for the alias target queue on the specified queue manager and fails because the alias target queue is not on the local queue manager.

Solution

Leave the **ObjectQmgrName** parameter blank, so that the clustering decides which queue manager to route to.

Messages are not arriving on the destination queues

Make sure that the corresponding cluster transmission queue is empty and also that the channel to the destination queue manager is running.

Symptom

Messages are not arriving on the destination queues.

Cause

The messages might be stuck at their origin queue manager.

Solution

1. Identify the transmission queue that is sending messages to the destination and the status of the channel.

```
1 : dis clusqmgr(QM1) CHANNEL(*) STATUS DEFTYPE QMTYPE XMITQ
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM1)     CLUSTER(DEMO)
CHANNEL(DEMO.QM1) DEFTYPE(CLUSSDRA)
QMTYPE(NORMAL)    STATUS(RUNNING)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.DEMO.QM1)
```

2. Make sure that the cluster transmission queue is empty.

```
1 : display ql(SYSTEM.CLUSTER.TRANSMIT.DEMO.QM1) curdepth
AMQ8409: Display Queue details.
QUEUE(SYSTEM.CLUSTER.TRANSMIT.DEMO.QM1) CURDEPTH(0)
```

Messages put to a cluster alias queue go to `SYSTEM.DEAD.LETTER.QUEUE`

A cluster alias queue resolves to a local queue that does not exist.

Symptom

Messages put to an alias queue go to `SYSTEM.DEAD.LETTER.QUEUE` with reason MQRC_UNKNOWN_ALIAS_BASE_Q.

Cause

A message is routed to a queue manager where a clustered alias queue is defined. A local target queue is not defined on that queue manager. Because the message was put with the `MQOO_BIND_ON_OPEN` open option, the queue manager cannot requeue the message.

When `MQOO_BIND_ON_OPEN` is used, the cluster queue alias is firmly bound. The resolved name is the name of the target queue and any queue manager on which the cluster queue alias is defined. The queue manager name is placed in the transmission queue header. If the target queue does not exist on the queue manager to which the message is sent, the message is put on the dead letter queue. The destination is not recomputed, because the transmission header contains the name of the target queue manager resolved by `MQOO_BIND_ON_OPEN`. If the alias queue had been opened with `MQOO_BIND_NOT_FIXED`, then the transmission queue header would contain a blank queue manager name, and the destination would be recomputed. In which case, if the local queue is defined elsewhere in the cluster, the message would be sent there.

Solution

1. Change all alias queue definitions to specify `DEFBIND ( NOTFIXED)`.
2. Use `MQOO_BIND_NOT_FIXED` as an open option when the queue is opened.
3. If you specify `MQOO_BIND_ON_OPEN`, ensure that a cluster alias that resolves to a local queue defined on the same queue manager as the alias.

A queue manager has out of date information about queues and channels in the cluster

Symptom

DISPLAY QCLUSTER and DISPLAY CLUSQMGR show objects which are out of date.

Cause

Updates to the cluster only flow between the full repositories over manually defined CLUSSDR channels. After the cluster has formed CLUSSDR channels display as DEFTYPE (CLUSSDRB) channels because they are both manual and automatic channels. There must be enough CLUSSDR channels to form a complete network between all the full repositories.

Solution

- Check that the queue manager where the object exists and the local queue manager are still connected to the cluster.
- Check that each queue manager can display all the full repositories in the cluster.
- Check whether the CLUSSDR channels to the full repositories are continually trying to restart.
- Check that the full repositories have enough CLUSSDR channels defined to correctly connect them together.

```
1 : dis clusqmgr(QM1) CHANNEL(*) STATUS DEFTYPE QMTYPE
XMITQ
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM1)     CLUSTER(DEMO)
CHANNEL(DEMO.QM1) DEFTYPE(CLUSSDRA)
QMTYPE(NORMAL)    STATUS(RUNNING)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.DEMO.QM1)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM2)     CLUSTER(DEMO)
CHANNEL(DEMO.QM2) DEFTYPE(CLUSRCVR)
QMTYPE(REPOS)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.DEMO.QM2)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM3)     CLUSTER(DEMO)
CHANNEL(DEMO.QM3) DEFTYPE(CLUSSDRB)
QMTYPE(REPOS)     STATUS(RUNNING)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.DEMO.QM3)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM4)     CLUSTER(DEMO)
CHANNEL(DEMO.QM4) DEFTYPE(CLUSSDRA)
QMTYPE(NORMAL)    STATUS(RUNNING)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.DEMO.QM4)
```

No changes in the cluster are being reflected in the local queue manager

The repository manager process is not processing repository commands, possibly because of a problem with receiving or processing messages in the command queue.

Symptom

No changes in the cluster are being reflected in the local queue manager.

Cause

The repository manager process is not processing repository commands.

Solution

1. Check that the SYSTEM.CLUSTER.COMMAND.QUEUE is empty.

   ```
   1 : display ql(SYSTEM.CLUSTER.COMMAND.QUEUE) curdepth
   AMQ8409: Display Queue details.
   QUEUE(SYSTEM.CLUSTER.COMMAND.QUEUE) CURDEPTH(0)
   ```

2. z/OS Check that the channel initiator is running on z/OS.
3. Check that there are no error messages in the error logs indicating the queue manager has a temporary resource shortage.

DISPLAY CLUSQMGR displays a queue manager twice

Use the RESET CLUSTER command to remove all traces of an old instance of a queue manager.

```
1 : display clusqmgr(QM1) qmid
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM1)                          CLUSTER(DEMO)
CHANNEL(DEMO.QM1)                      QMID(QM1_2002-03-04_11.07.01)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM1)                          CLUSTER(DEMO)
CHANNEL(DEMO.QM1)                      QMID(QM1_2002-03-04_11.04.19)
```

The cluster functions correctly with the older version of the queue manager being ignored. After about 90 days, the cluster's knowledge of the older version of the queue manager expires, and is deleted automatically. However you might prefer to delete this information manually.

Cause

1. The queue manager might have been deleted and then re-created and redefined.
2. It might have been cold-started on z/OS, without first following the procedure to remove a queue manager from a cluster.

Solution

To remove all trace of the queue manager immediately use the RESET CLUSTER command from a full repository queue manager. The command removes the older unwanted queue manager and its queues from the cluster.

```
2 : reset cluster(DEMO) qmid('QM1_2002-03-04_11.04.19') action(FORCEREMOVE) queues(yes)
AMQ8559: RESET CLUSTER accepted.
```

Using the RESET CLUSTER command stops auto-defined cluster sender channels for the affected queue manager. You must manually restart any cluster sender channels that are stopped, after completing the RESET CLUSTER command.

A queue manager does not rejoin the cluster

After issuing a RESET or REFRESH cluster command the channel from the queue manager to the cluster might be stopped. Check the cluster channel status and restart the channel.

Symptom

A queue manager does not rejoin a cluster after issuing the RESET CLUSTER and REFRESH CLUSTER commands.

Cause

A side effect of the RESET and REFRESH commands might be that a channel is stopped. A channel is stopped in order that the correct version of the channel runs when RESET or REFRESH command is completed.

Solution

Check that the channels between the problem queue manager and the full repositories are running and use the START CHANNEL command if necessary.

Related information:

Clustering: Using REFRESH CLUSTER best practices

Workload balancing set on a cluster-sender channel is not working

Any workload balancing you specify on a cluster-sender channel is likely to be ignored. Instead, specify the cluster workload channel attributes on the cluster-receiver channel at the target queue manager.

Symptom

You have specified one or more cluster workload channel attributes on a cluster-sender channel. The resulting workload balancing is not as you were expecting.

Cause

Any workload balancing you specify on a cluster-sender channel is likely to be ignored. For an explanation of this, see Cluster channels. Note that you still get some form of workload balancing, based either on cluster defaults or on properties set on the matching cluster-receiver channel at the target queue manager.

Solution

Specify the cluster workload channel attributes on the cluster-receiver channel at the target queue manager.

Related information:
CLWLPRTY channel attribute
CLWLRANK channel attribute
CLWLWGHT channel attribute
NETPRTY channel attribute

Out of date information in a restored cluster

After restoring a queue manager, its cluster information is out of date. Refresh the cluster information with the **REFRESH CLUSTER** command.

Problem

After an image backup of QM1, a partial repository in cluster DEMO has been restored and the cluster information it contains is out of date.

Solution

On QM1, issue the command REFRESH CLUSTER(DEMO).

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

When you run REFRESH CLUSTER(DEMO) on QM1, you remove all the information QM1 has about the cluster DEMO, except for QM1's knowledge of itself and its own queues, and of how to access the full repositories in the cluster. QM1 then contacts the full repositories, and tells them about itself and its queues. QM1 is a partial repository, so the full repositories don't immediately tell QM1 about all the other partial repositories in the cluster. Instead, QM1 slowly builds up its knowledge of the other partial repositories through information it receives as and when each of the other queues and queue managers is next active in the cluster.

Cluster queue manager force removed from a full repository by mistake

Restore the queue manager to the full repository by issuing the command **REFRESH CLUSTER** on the queue manager that was removed from the repository.

Problem

The command, RESET CLUSTER(DEMO) QMNAME(QM1) ACTION(FORCEREMOVE) was issued on a full repository in cluster DEMO by mistake.

Solution

On QM1, issue the command REFRESH CLUSTER(DEMO).

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

Possible repository messages deleted

Messages destined for a queue manager were removed from the SYSTEM.CLUSTER.TRANSMIT.QUEUE in other queue managers. Restore the information by issuing the REFRESH CLUSTER command on the affected queue manager.

Problem

Messages destined for QM1 were removed from the SYSTEM.CLUSTER.TRANSMIT.QUEUE in other queue managers and they might have been repository messages.

Solution

On QM1, issue the command REFRESH CLUSTER(DEMO).

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

QM1 removes all information it has about the cluster DEMO, except that relating to the cluster queue managers which are the full repositories in the cluster. Assuming that this information is still correct, QM1 contacts the full repositories. QM1 informs the full repositories about itself and its queues. It recovers the information for queues and queue managers that exist elsewhere in the cluster as they are opened.

Two full repositories moved at the same time

If you move both full repositories to new network addresses at the same time, the cluster is not updated with the new addresses automatically. Follow the procedure to transfer the new network addresses. Move the repositories one at a time to avoid the problem.

Problem

Cluster DEMO contains two full repositories, QM1 and QM2. They were both moved to a new location on the network at the same time.

Solution

1. Alter the CONNAME in the CLUSRCVR and CLUSSDR channels to specify the new network addresses.
2. Alter one of the queue managers (QM1 or QM2) so it is no longer a full repository for any cluster.
3. On the altered queue manager, issue the command REFRESH CLUSTER(*) REPOS(YES).

   Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.
4. Alter the queue manager so it is acting as a full repository.

Recommendation

You could avoid the problem as follows:

1. Move one of the queue managers, for example QM2, to its new network address.
2. Alter the network address in the QM2 CLUSRCVR channel.
3. Start the QM2 CLUSRCVR channel.
4. Wait for the other full repository queue manager, QM1, to learn the new address of QM2.
5. Move the other full repository queue manager, QM1, to its new network address.

6. Alter the network address in the QM1 CLUSRCVR channel.
7. Start the QM1 CLUSRCVR channel.
8. Alter the manually defined CLUSSDR channels for the sake of clarity, although at this stage they are not needed for the correct operation of the cluster.

The procedure forces QM2 to reuse the information from the correct CLUSSDR channel to re-establish contact with QM1 and then rebuild its knowledge of the cluster. Additionally, having once again contacted QM1, it is given its own correct network address based on the CONNAME in QM2 CLUSRCVR definition.

Unknown state of a cluster

Restore the cluster information in all the full repositories to a known state by rebuilding the full repositories from all the partial repositories in the cluster.

Problem

Under normal conditions the full repositories exchange information about the queues and queue managers in the cluster. If one full repository is refreshed, the cluster information is recovered from the other.

The problem is how to completely reset all the systems in the cluster to restore a known state to the cluster.

Solution

To stop cluster information being updated from the unknown state of the full repositories, all the CLUSRCVR channels to full repositories are stopped. The CLUSSDR channels change to inactive.

When you refresh the full repository systems, none of them are able to communicate, so they start from the same cleared state.

When you refresh the partial repository systems, they rejoin the cluster and rebuild it to the complete set of queue managers and queues. The cluster information in the rebuilt full is restored to a known state.

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

1. On all the full repository queue managers, follow these steps:
   a. Alter queue managers that are full repositories so they are no longer full repositories.
   b. Resolve any in doubt CLUSSDR channels.
   c. Wait for the CLUSSDR channels to become inactive.
   d. Stop the CLUSRCVR channels.
   e. When all the CLUSRCVR channels on all the full repository systems are stopped, issue the command REFRESH CLUSTER(DEMO) REPOS(YES).
   f. Alter the queue managers so they are full repositories.
   g. Start the CLUSRCVR channels to re-enable them for communication.
2. On all the partial repository queue managers, follow these steps:
   a. Resolve any in doubt CLUSSDR channels.
   b. Make sure all CLUSSDR channels on the queue manager are stopped or inactive.
   c. Issue the command REFRESH CLUSTER(DEMO) REPOS(YES).

What happens when a cluster queue manager fails

When a cluster queue manager fails, some undelivered messages are sent to other queue managers in the cluster. Messages that are in-flight wait until the queue manager is restarted. Use a high-availability mechanism to restart a queue manager automatically.

Problem

If a message-batch is sent to a particular queue manager and that queue manager becomes unavailable, what happens at the sending queue manager?

Explanation

Except for non-persistent messages on an NPMSPEED(FAST) channel, the undelivered batch of messages is backed out to the cluster transmission queue on the sending queue manager. On an NPMSPEED(FAST) channel, non-persistent messages are not batched, and one might be lost.

- Indoubt messages, and messages that are bound to the unavailable queue manager, wait until the queue manager becomes available again.
- Other messages are delivered to alternative queue managers selected by the workload management routine.

Solution

The unavailable cluster queue manager can be restarted automatically, either by being configured as a multi-instance queue manager, or by a platform-specific high availability mechanism.

What happens when a repository fails

How you know a repository has failed and what to do to fix it?

Problem

1. Cluster information is sent to repositories (whether full or partial) on a local queue called `SYSTEM.CLUSTER.COMMAND.QUEUE`. If this queue fills up, perhaps because the queue manager has stopped working, the cluster-information messages are routed to the dead-letter queue.
2. The repository runs out of storage.

Solution

1. Monitor the messages on your queue manager log z/OS or z/OS system console to detect if `SYSTEM.CLUSTER.COMMAND.QUEUE` is filling up. If it is, you need to run an application to retrieve the messages from the dead-letter queue and reroute them to the correct destination.
2. If errors occur on a repository queue manager, messages tell you what error has occurred and how long the queue manager waits before trying to restart.
   - z/OS On IBM MQ for z/OS, the `SYSTEM.CLUSTER.COMMAND.QUEUE` is disabled for MQGET.
   - When you have identified and resolved the error, enable the `SYSTEM.CLUSTER.COMMAND.QUEUE` so that the queue manager can restart successfully.
3. In the unlikely event of the repository running out of storage, storage allocation errors are sent to the queue-manager log z/OS or z/OS system console. To fix the storage problem, stop and then restart the queue manager. When the queue manager is restarted, more storage is automatically allocated to hold all the repository information.

What happens if a cluster queue is disabled for MQPUT

All instances of a cluster queue that is being used for workload balancing might be disabled for MQPUT. Applications putting a message to the queue either receive a `MQRC_CLUSTER_PUT_INHIBITED` or a `MQRC_PUT_INHIBITED` return code . You might want to modify this behavior.

Problem

When a cluster queue is disabled for MQPUT, its status is reflected in the repository of each queue manager that is interested in that queue. The workload management algorithm tries to send messages to destinations that are enabled for MQPUT. If there are no destinations enabled for MQPUT and no local instance of a queue, an MQOPEN call that specified `MQOO_BIND_ON_OPEN` returns a return code of `MQRC_CLUSTER_PUT_INHIBITED` to the application. If `MQOO_BIND_NOT_FIXED` is specified, or there is a local instance of the queue, an MQOPEN call succeeds but subsequent MQPUT calls fail with return code `MQRC_PUT_INHIBITED`.

Solution

You can write a user exit program to modify the workload management routines so that messages can be routed to a destination that is disabled for MQPUT.

A message can arrive at a destination that is disabled for MQPUT. The message might have been in flight at the time the queue became disabled, or a workload exit might have chosen the destination explicitly. The workload management routine at the destination queue manager has a number of ways to deal with the message:

- Choose another appropriate destination, if there is one.
- Place the message on the dead-letter queue.
- Return the message to the originator, if there is no dead-letter queue

Queue managers troubleshooting

Use the advice given here to help you to resolve common problems that can arise when you use queue managers.

Queue manager unavailable error

- **Scenario:** You receive a *queue manager unavailable* error.
- **Explanation:** Configuration file errors typically prevent queue managers from being found, and result in *queue manager unavailable* errors. On Windows, problems in the qm.ini file can cause *queue manager unavailable* errors when a queue manager is started.
- **Solution:** Ensure that the configuration files exist, and that the IBM MQ configuration file references the correct queue manager and log directories. On Windows, check for problems in the qm.ini file.

Undelivered messages troubleshooting

Use the advice given here to help you to resolve problems when messages do are not delivered successfully.

- **Scenario:** Messages do not arrive on a queue when you are expecting them.
- **Explanation:** Messages that cannot be delivered for some reason are placed on the dead-letter queue.
- **Solution:** You can check whether the queue contains any messages by issuing an MQSC DISPLAY QUEUE command.

  If the queue contains messages, you can use the provided browse sample application (amqsbcg) to browse messages on the queue using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.

You must decide how to dispose of any messages found on the dead-letter queue, depending on the reasons for the messages being put on the queue. Problems might occur if you do not associate a dead-letter queue with each queue manager.

For more information about dead-letter queues and handling undelivered messages, see Working with dead-letter queues.

TLS/SSL troubleshooting information

Use the information listed here to help you solve problems with your TLS/SSL system.

Overview

For the error caused by *Using non-FIPS cipher with FIPS enabled on client*, you receive the following error message:

JMSCMQ001

IBM MQ call failed with completion code *2 ('MQCC_FAILED')* reason *2397 ('MQRC_JSSE_ERROR')*

For every other problem documented within this topic you receive either the previous error message, or the following error message, or both:

JMSWMQ0018

Failed to connect to queue manager *'queue-manager-name'* with connection mode *'connection-mode'* and host name *'host-name'*

For each problem documented within this topic, the following information is provided:
- Output from the sample `SystemOut.log` or `Console`, detailing the cause of the exception..
- Queue manager error log information.
- Solution to the problem.

Note:
- You should always list out the stacks and the cause of the first exception.
- Whether or not the error information is written to the `stdout` log file depends on how the application is written, and on which framework you are using.
- The sample code includes stacks and line numbers. This information is useful guidance, but the stacks and line numbers are likely to change from one fix pack to another. You should use the stacks and line numbers as a guide to locating the correct section, and not use the information specifically for diagnostic purposes.

Cipher suite not set on client

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9641: Remote CipherSpec error for channel
'SYSTEM.DEF.SVRCONN' to host ''. [3=SYSTEM.DEF.SVRCONN]
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.analyseErrorSegment(RemoteConnection.java:4176)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.receiveTSH(RemoteConnection.java:2969)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.initSess(RemoteConnection.java:1180)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:838)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
```

```
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs
AMQ9639: Remote channel *'SYSTEM.DEF.SVRCONN'* did not specify a CipherSpec.

Solution
Set a CipherSuite on the client so that both ends of the channel have a matching CipherSuite or CipherSpec pair.

Cipher suite not set on server

Output
Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9641: Remote CipherSpec error
for channel 'SYSTEM.DEF.SVRCONN' to host ''. [3=SYSTEM.DEF.SVRCONN]
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.analyseErrorSegment(RemoteConnection.java:4176)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.receiveTSH(RemoteConnection.java:2969)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.initSess(RemoteConnection.java:1180)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:838)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs
AMQ9639: Remote channel *'SYSTEM.DEF.SVRCONN'* did not specify a CipherSpec.

Solution
Change channel *'SYSTEM.DEF.SVRCONN'* to specify a valid CipherSpec.

Cipher Mismatch

Output
Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9641: Remote CipherSpec error
for channel 'SYSTEM.DEF.SVRCONN' to host ''. [3=SYSTEM.DEF.SVRCONN]
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.analyseErrorSegment(RemoteConnection.java:4176)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.receiveTSH(RemoteConnection.java:2969)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.initSess(RemoteConnection.java:1180)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:838)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs
AMQ9631: The CipherSpec negotiated during the SSL handshake does not match the required CipherSpec for channel *'SYSTEM.DEF.SVRCONN'*.

Solution
Change either the SSLCIPH definition of the server-connection channel or the Cipher suite of the client so that the two ends have a matching CipherSuite or CipherSpec pair.

Missing client personal certificate

Output
Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9503: Channel negotiation failed. [3=SYSTEM.DEF.SVRCONN]
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.analyseErrorSegment(RemoteConnection.java:4176)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.receiveTSH(RemoteConnection.java:2969)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.initSess(RemoteConnection.java:1180)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:838)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs
: AMQ9637: Channel is lacking a certificate.

Solution
: Ensure that the key database of the queue manager contains a signed personal certificate from the truststore of the client.

Missing server personal certificate

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
[1=javax.net.ssl.SSLHandshakeException[Remote host closed connection during handshake],
3=localhost/127.0.0.1:1418 (localhost),4=SSLSocket.startHandshake,5=default]
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1173)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:835)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
... 12 more
```

Caused by:

```
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
at com.ibm.jsse2.qc.a(qc.java:158)
at com.ibm.jsse2.qc.h(qc.java:185)
at com.ibm.jsse2.qc.a(qc.java:566)
at com.ibm.jsse2.qc.startHandshake(qc.java:120)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1142)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1134)
at java.security.AccessController.doPrivileged(AccessController.java:229)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1134)
... 17 more
```

Caused by:

```
java.io.EOFException: SSL peer shut down incorrectly
at com.ibm.jsse2.a.a(a.java:19)
at com.ibm.jsse2.qc.a(qc.java:207)
```

Queue manager error logs
: AMQ9637: Channel is lacking a certificate.

Solution
: Ensure that the key database of the queue manager contains a signed personal certificate from the truststore of the client.

Missing server signer on client

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
[1=javax.net.ssl.SSLHandshakeException[com.ibm.jsse2.util.j:
PKIX path validation failed: java.security.cert.CertPathValidatorException:
The certificate issued by CN=JohnDoe, O=COMPANY, L=YOURSITE, C=XX is not trusted; internal cause is:
java.security.cert.CertPathValidatorException: Signature does not match.],3=localhost/127.0.0.1:1418
(localhost),4=SSLSocket.startHandshake,5=default]
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1173)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:835)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
```

```
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
...
```

Caused by:

```
javax.net.ssl.SSLHandshakeException: com.ibm.jsse2.util.j: PKIX path validation failed:
java.security.cert.CertPathValidatorException:
The certificate issued by CN=JohnDoe, O=COMPANY, L=YOURSITE, C=XX is not trusted;
internal cause is: java.security.cert.CertPathValidatorException: Signature does not match.
...
```

Caused by:

```
com.ibm.jsse2.util.j: PKIX path validation failed: java.security.cert.CertPathValidatorException:
The certificate issued by CN=JohnDoe, O=COMPANY, L=YOURSITE, C=XX is not trusted;
internal cause is: java.security.cert.CertPathValidatorException: Signature does not match.
at com.ibm.jsse2.util.h.a(h.java:99)
at com.ibm.jsse2.util.h.b(h.java:27)
at com.ibm.jsse2.util.g.a(g.java:14)
at com.ibm.jsse2.yc.a(yc.java:68)
at com.ibm.jsse2.yc.a(yc.java:17)
at com.ibm.jsse2.yc.checkServerTrusted(yc.java:154)
at com.ibm.jsse2.bb.a(bb.java:246)
... 28 more
```

Caused by:

```
java.security.cert.CertPathValidatorException:
The certificate issued by CN=JohnDoe, O=COMPANY, L=YOURSITE, C=XX is not trusted;
internal cause is: java.security.cert.CertPathValidatorException: Signature does not match.
at com.ibm.security.cert.BasicChecker.(BasicChecker.java:111)
at com.ibm.security.cert.PKIXCertPathValidatorImpl.engineValidate(PKIXCertPathValidatorImpl.java:174)
at java.security.cert.CertPathValidator.validate(CertPathValidator.java:265)
at com.ibm.jsse2.util.h.a(h.java:13)
... 34 more
```

Caused by:

```
java.security.cert.CertPathValidatorException: Signature does not match.
at com.ibm.security.cert.CertPathUtil.findIssuer(CertPathUtil.java:297)
at com.ibm.security.cert.BasicChecker.(BasicChecker.java:108)
```

Queue manager error logs

AMQ9665: SSL connection closed by remote end of channel *'????'*.

Solution

Add the certificate used to sign the personal certificate of the queue manager to the truststore of the client.

Missing client signer on server

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
[1=java.net.SocketException[Software caused connection abort: socket write error],
3=localhost/127.0.0.1:1418 (localhost),4=SSLSocket.startHandshake,5=default]
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1173)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:835)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
... 12 more
```

Caused by:

```
java.net.SocketException: Software caused connection abort: socket write error
at java.net.SocketOutputStream.socketWrite(SocketOutputStream.java:120)
at java.net.SocketOutputStream.write(SocketOutputStream.java:164)
at com.ibm.jsse2.c.a(c.java:57)
at com.ibm.jsse2.c.a(c.java:34)
at com.ibm.jsse2.qc.b(qc.java:527)
at com.ibm.jsse2.qc.a(qc.java:635)
at com.ibm.jsse2.qc.a(qc.java:743)
at com.ibm.jsse2.ab.a(ab.java:550)
at com.ibm.jsse2.bb.b(bb.java:194)
at com.ibm.jsse2.bb.a(bb.java:162)
at com.ibm.jsse2.bb.a(bb.java:7)
at com.ibm.jsse2.ab.r(ab.java:529)
at com.ibm.jsse2.ab.a(ab.java:332)
at com.ibm.jsse2.qc.a(qc.java:435)
at com.ibm.jsse2.qc.h(qc.java:185)
at com.ibm.jsse2.qc.a(qc.java:566)
at com.ibm.jsse2.qc.startHandshake(qc.java:120)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1142)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1134)
at java.security.AccessController.doPrivileged(AccessController.java:229)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1134)
```

Queue manager error logs

AMQ9633: Bad SSL certificate for channel *'????'*.

Solution

Add the certificate used to sign the personal certificate of the client to the key database of the queue manager.

SSLPEER set on server does not match certificate

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9643: Remote SSL peer name error for channel
'SYSTEM.DEF.SVRCONN' on host ''. [3=SYSTEM.DEF.SVRCONN]
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.analyseErrorSegment(RemoteConnection.java:4176)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.receiveTSH(RemoteConnection.java:2969)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.initSess(RemoteConnection.java:1180)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:838)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs

AMQ9636: SSL distinguished name does not match peer name, channel *'SYSTEM.DEF.SVRCONN'*.

Solution

Ensure the value of SSLPEER set on the server-connection channel matches the distinguished name of the certificate.

SSLPEER set on client does not match certificate

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2398;AMQ9636: SSL distinguished name does not match peer name,
channel '?'. [CN=JohnDoe, O=COMPANY, L=YOURSITE, C=XX]
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1215)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:835)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs

AMQ9208: Error on receive from host *host-name (address).*

Solution

Ensure the value of SSLPEER set in the client matches the distinguished name of the certificate.

Using a non-FIPS cipher with FIPS enabled on client

Output

```
Check the queue manager is started and if running in client mode, check there is a listener running.
Please see the linked exception for more information.
at com.ibm.msg.client.wmq.common.internal.Reason.reasonToException(Reason.java:578)
at com.ibm.msg.client.wmq.common.internal.Reason.createException(Reason.java:214)
at com.ibm.msg.client.wmq.internal.WMQConnection.getConnectOptions(WMQConnection.java:1423)
at com.ibm.msg.client.wmq.internal.WMQConnection.(WMQConnection.java:339)
at com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection
(WMQConnectionFactory.java:6865)
at com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createProviderConnection
(WMQConnectionFactory.java:6221)
at com.ibm.msg.client.jms.admin.JmsConnectionFactoryImpl._createConnection
(JmsConnectionFactoryImpl.java:285)
at com.ibm.msg.client.jms.admin.JmsConnectionFactoryImpl.createConnection
(JmsConnectionFactoryImpl.java:233)
at com.ibm.mq.jms.MQConnectionFactory.createCommonConnection(MQConnectionFactory.java:6016)
at com.ibm.mq.jms.MQConnectionFactory.createConnection(MQConnectionFactory.java:6041)
at tests.SimpleSSLConn.runTest(SimpleSSLConn.java:46)
at tests.SimpleSSLConn.main(SimpleSSLConn.java:26)
```

Caused by:

```
com.ibm.mq.MQException: JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED')
reason '2400' ('MQRC_UNSUPPORTED_CIPHER_SUITE').
at com.ibm.msg.client.wmq.common.internal.Reason.createException(Reason.java:202)
```

Queue manager error logs

Not applicable.

Solution

Use a FIPS-enabled cipher, or disable FIPS on the client.

Using a non-FIPS cipher with FIPS enabled on the queue manager

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
[1=javax.net.ssl.SSLHandshakeException[Received fatal alert: handshake_failure],
3=localhost/127.0.0.1:1418 (localhost),4=SSLSocket.startHandshake,5=default]
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1173)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:835)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
... 12 more
```

Caused by:

```
javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure
at com.ibm.jsse2.j.a(j.java:13)
at com.ibm.jsse2.j.a(j.java:18)
at com.ibm.jsse2.qc.b(qc.java:601)
at com.ibm.jsse2.qc.a(qc.java:100)
at com.ibm.jsse2.qc.h(qc.java:185)
at com.ibm.jsse2.qc.a(qc.java:566)
at com.ibm.jsse2.qc.startHandshake(qc.java:120)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1142)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1134)
at java.security.AccessController.doPrivileged(AccessController.java:229)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1134)
```

Queue manager error logs
AMQ9616: The CipherSpec proposed is not enabled on the server.

Solution
Use a FIPS-enabled cipher, or disable FIPS on the queue manager.

Can not find client keystore using IBM JRE

Output
Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9204: Connection to host 'localhost(1418)' rejected.
[1=com.ibm.mq.jmqi.JmqiException[CC=2;RC=2059;AMQ9503: Channel negotiation failed.
[3=SYSTEM.DEF.SVRCONN]],3=localhost(1418),5=RemoteConnection.analyseErrorSegment]
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:2450)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1396)
at com.ibm.mq.ese.jmqi.InterceptedJmqiImpl.jmqiConnect(InterceptedJmqiImpl.java:376)
at com.ibm.mq.ese.jmqi.ESEJMQI.jmqiConnect(ESEJMQI.java:561)
at com.ibm.msg.client.wmq.internal.WMQConnection.(WMQConnection.java:342)
... 8 more
```

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9503: Channel negotiation failed. [3=SYSTEM.DEF.SVRCONN]
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.analyseErrorSegment(RemoteConnection.java:4176)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.receiveTSH(RemoteConnection.java:2969)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.initSess(RemoteConnection.java:1180)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:838)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs
AMQ9637: Channel is lacking a certificate.

Solution
Ensure the JVM property `javax.net.ssl.keyStore` specifies the location of a valid keystore.

Can not find client keystore using Oracle JRE

Output
Caused by:

```
java.security.PrivilegedActionException: java.io.FileNotFoundException:
C:\<filepath>\wrongkey.jks (The system cannot find the file specified)
at java.security.AccessController.doPrivileged(Native Method)
at sun.security.ssl.SSLContextImpl$DefaultSSLContext.getDefaultKeyManager(Unknown Source)
at sun.security.ssl.SSLContextImpl$DefaultSSLContext.(Unknown Source)
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Unknown Source)
at java.lang.reflect.Constructor.newInstance(Unknown Source)
at java.lang.Class.newInstance0(Unknown Source)
at java.lang.Class.newInstance(Unknown Source)
... 28 more
```

Caused by:

```
java.io.FileNotFoundException: C:\<filepath>\wrongkey.jks (The system cannot find the file specified)
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.(Unknown Source)
at java.io.FileInputStream.(Unknown Source)
at sun.security.ssl.SSLContextImpl$DefaultSSLContext$2.run(Unknown Source)
at sun.security.ssl.SSLContextImpl$DefaultSSLContext$2.run(Unknown Source)
```

Queue manager error logs
AMQ9637: Channel is lacking a certificate.

Solution
Ensure the JVM property `javax.net.ssl.keyStore` specifies the location of a valid keystore.

Keystore password error - IBM JRE

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9503: Channel negotiation failed. [3=SYSTEM.DEF.SVRCONN]
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.analyseErrorSegment(RemoteConnection.java:4176)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.receiveTSH(RemoteConnection.java:2969)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.initSess(RemoteConnection.java:1180)
at com.ibm.mq.jmqi.remote.impl.RemoteConnection.connect(RemoteConnection.java:838)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSessionFromNewConnection
(RemoteConnectionSpecification.java:409)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionSpecification.getSession
(RemoteConnectionSpecification.java:305)
at com.ibm.mq.jmqi.remote.impl.RemoteConnectionPool.getSession(RemoteConnectionPool.java:146)
at com.ibm.mq.jmqi.remote.api.RemoteFAP.jmqiConnect(RemoteFAP.java:1868)
```

Queue manager error logs

AMQ9637: Channel is lacking a certificate.

Solution

Ensure that the value of the JVM property `javax.net.ssl.keyStorePassword` specifies the password for the keystore specified by `javax.net.ssl.keyStore`.

Truststore password error - IBM JRE

Output

Caused by:

```
javax.net.ssl.SSLHandshakeException: java.security.cert.CertificateException:
No X509TrustManager implementation available
at com.ibm.jsse2.j.a(j.java:13)
at com.ibm.jsse2.qc.a(qc.java:204)
at com.ibm.jsse2.ab.a(ab.java:342)
at com.ibm.jsse2.ab.a(ab.java:222)
at com.ibm.jsse2.bb.a(bb.java:157)
at com.ibm.jsse2.bb.a(bb.java:492)
at com.ibm.jsse2.ab.r(ab.java:529)
at com.ibm.jsse2.ab.a(ab.java:332)
at com.ibm.jsse2.qc.a(qc.java:435)
at com.ibm.jsse2.qc.h(qc.java:185)
at com.ibm.jsse2.qc.a(qc.java:566)
at com.ibm.jsse2.qc.startHandshake(qc.java:120)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1142)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1134)
at java.security.AccessController.doPrivileged(AccessController.java:229)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1134)
... 17 more
```

Caused by:

```
java.security.cert.CertificateException: No X509TrustManager implementation available
at com.ibm.jsse2.xc.checkServerTrusted(xc.java:2)
at com.ibm.jsse2.bb.a(bb.java:246)
```

Queue manager error logs

AMQ9665: SSL connection closed by remote end of channel `'????'`.

Solution

Ensure that the value of the JVM property `javax.net.ssl.trustStorePassword` specifies the password for the keystore specified by `javax.net.ssl.trustStore`.

Can not find or open queue manager key database

Output

Caused by:

```
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
at com.ibm.jsse2.qc.a(qc.java:158)
at com.ibm.jsse2.qc.h(qc.java:185)
at com.ibm.jsse2.qc.a(qc.java:566)
at com.ibm.jsse2.qc.startHandshake(qc.java:120)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1142)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1134)
at java.security.AccessController.doPrivileged(AccessController.java:229)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1134)
... 17 more
```

Caused by:

```
java.io.EOFException: SSL peer shut down incorrectly
at com.ibm.jsse2.a.a(a.java:19)
at com.ibm.jsse2.qc.a(qc.java:207)
```

Queue manager error logs

AMQ9657: The key repository could not be opened (channel '????').

Solution

Ensure that the key repository you specify exists and that its permissions are such that the IBM MQ process involved can read from it.

Can not find or use queue manager key database password stash file

Output

Caused by:

```
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
at com.ibm.jsse2.qc.a(qc.java:158)
at com.ibm.jsse2.qc.h(qc.java:185)
at com.ibm.jsse2.qc.a(qc.java:566)
at com.ibm.jsse2.qc.startHandshake(qc.java:120)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1142)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection$6.run(RemoteTCPConnection.java:1134)
at java.security.AccessController.doPrivileged(AccessController.java:229)
at com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection.protocolConnect(RemoteTCPConnection.java:1134)
... 17 more
```

Caused by:

```
ava.io.EOFException: SSL peer shut down incorrectly
at com.ibm.jsse2.a.a(a.java:19)
at com.ibm.jsse2.qc.a(qc.java:207)
```

Queue manager error logs

AMQ9660: SSL key repository: password stash file absent or unusable.

Solution

Ensure that a password stash file has been associated with the key database file in the same directory, and that the user ID, under which IBM MQ is running, has read access to both files.

IBM MQ Telemetry troubleshooting

Look for a troubleshooting task to help you solve a problem with running IBM MQ Telemetry applications.

Related information:
IBM MQ Telemetry

Location of telemetry logs, error logs, and configuration files

Find the logs, error logs, and configuration files used by IBM MQ Telemetry.

Note: The examples are coded for Windows systems. Change the syntax to run the examples on AIX or Linux systems.

Server-side logs

The installation wizard for IBM MQ Telemetry writes messages to its installation log:

```
WMQ program directory\mqxr
```

The telemetry (MQXR) service writes messages to the IBM MQ queue manager error log, and FDC files to the IBM MQ error directory:

```
 WMQ data directory\Qmgrs\qMgrName\errors\AMQERR01.LOG
WMQ data directory\errors\AMQ nnn.n.FDC
```

It also writes a log for the telemetry (MQXR) service. The log displays the properties the service started with, and errors it has found acting as a proxy for an MQTT client. For example, unsubscribing from a subscription that the client did not create. The log path is:

```
WMQ data directory\Qmgrs\qMgrName\errors\mqxr.log
```

The IBM MQ telemetry sample configuration created by IBM MQ explorer starts the telemetry (MQXR) service using the command **`runMQXRService`**, which is in *`WMQ Telemetry install directory`*`\bin`. This command writes to:

```
 WMQ data directory\Qmgrs\qMgrName\mqxr.stdout
WMQ data directory\Qmgrs\qMgrName\mqxr.stderr
```

Modify **`runMQXRService`** to display the paths configured for the telemetry (MQXR) service, or to echo the initialization before starting the telemetry (MQXR) service.

Server-side configuration files

Telemetry channels and telemetry (MQXR) service

Restriction: The format, location, content, and interpretation of the telemetry channel configuration file might change in future releases. You must use IBM MQ Explorer to configure telemetry channels.

IBM MQ Explorer saves telemetry configurations in the `mqxr_win.properties` file on Windows systems, and the `mqxr_unix.properties` file on AIX or Linux systems. The properties files are saved in the telemetry configuration directory:

```
WMQ data directory\Qmgrs\qMgrName\mqxr
```

Figure 138. Telemetry configuration directory on Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Figure 139. Telemetry configuration directory on AIX or Linux

JVM Set Java properties that are passed as arguments to the telemetry (MQXR) service in the file, `java.properties`. The properties in the file are passed directly to the JVM running the telemetry (MQXR) service. They are passed as additional JVM properties on the Java command line. Properties set on the command line take precedence over properties added to the command line from the `java.properties` file.

Find the `java.properties` file in the same folder as the telemetry configurations. See Figure 138 on page 1829 and Figure 139.

Modify `java.properties` by specifying each property as a separate line. Format each property exactly as you would to pass the property to the JVM as an argument. For example:

```
-Xmx1024m
-Xms1024m
```

JAAS The JAAS configuration file is described in Telemetry channel JAAS configuration, which includes the sample JAAS configuration file, JAAS.config, shipped with IBM MQ Telemetry.

If you configure JAAS, you are almost certainly going to write a class to authenticate users to replace the standard JAAS authentication procedures.

To include your Login class in the class path used by the telemetry (MQXR) service class path, provide an IBM MQ `service.env` configuration file.

Set the class path for your JAAS LoginModule in `service.env`. You cannot use the variable, `%classpath%` in `service.env`. The class path in `service.env` is added to the class path already set in the telemetry (MQXR) service definition.

Display the class paths that are being used by the telemetry (MQXR) service by adding `echo set classpath` to `runMQXRService.bat`. The output is sent to `mqxr.stdout`.

The default location for the `service.env` file is:

```
WMQ data directory\service.env
```

Override these settings with a `service.env` file for each queue manager in:

```
WMQ data directory\Qmgrs\qMgrName\service.env
```

```
CLASSPATH= WMQ Install Directory\mqxr\samples
```

Note: `service.env` must not contain any variables. Substitute the actual value of *`WMQ Install Directory`*.

Figure 140. Sample `service.env` for Windows.

A sample `service.env` file to use the sample `LoginModule.class`.

Trace See"Tracing the telemetry (MQXR) service" on page 1832. The parameters to configure trace are stored in two files:

```
  WMQ data directory\Qmgrs\qMgrName\mqxr\trace.config
 WMQ data directory\Qmgrs\qMgrName\mqxr\mqxrtrace.properties
```

Client-side log files

The default file persistence class in the Java SE MQTT client supplied with IBM WebSphere MQ Telemetry creates a folder with the name: *clientIdentifier*-**tcp***hostNameport* or *clientIdentifier*-**ssl***hostNameport* in the client working directory. The folder name tells you the `hostName` and `port` used in the connection attempt. The folder contains messages that have been stored by the persistence class. The messages are deleted when they have been delivered successfully.

The folder is deleted when a client, with a clean session, ends.

If client trace is turned on, the unformatted log is, by default, stored in the client working directory. The trace file is called `mqtt- n.trc`

Client-side configuration files

Set trace and SSL properties for the MQTT Java client using Java property files, or set the properties programmatically. Pass the properties to the MQTT Java client using the JVM `-D` switch: for example,

```
Java -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
-Dcom.ibm.ssl.keyStore=C:\\MyKeyStore.jks
```

See“Tracing the MQTT v3 Java client” on page 1833.For links to client API documentation for the MQTT client libraries, see MQTT client programming reference.

MQTT v3 Java client reason codes

Look up the causes of reason codes in an MQTT v3 Java client exception or throwable.

Table 174. MQTT v3 Java client reason codes

Reason code	Value	Cause
REASON_CODE_BROKER_UNAVAILABLE	3	
REASON_CODE_CLIENT_ALREADY_CONNECTED	32100	The client is already connected.
REASON_CODE_CLIENT_ALREADY_DISCONNECTED	32101	The client is already disconnected.
REASON_CODE_CLIENT_DISCONNECT_PROHIBITED	32107	Thrown when an attempt to call MqttClient.disconnect has been made from within a method on MqttCallback.
REASON_CODE_CLIENT_DISCONNECTING	32102	The client is currently disconnecting and cannot accept any new work.
REASON_CODE_CLIENT_EXCEPTION	0	Client encountered an exception.
REASON_CODE_CLIENT_NOT_CONNECTED	32104	The client is not connected to the server.
REASON_CODE_CLIENT_TIMEOUT	32000	Client timed out while waiting for a response from the server.
REASON_CODE_FAILED_AUTHENTICATION	4	Authentication with the server has failed, due to a bad user name or password.
REASON_CODE_INVALID_CLIENT_ID	2	The server has rejected the supplied client ID.
REASON_CODE_INVALID_PROTOCOL_VERSION	1	The protocol version requested is not supported by the server.
REASON_CODE_NO_MESSAGE_IDS_AVAILABLE	32001	Internal error, caused by no new message IDs being available.
REASON_CODE_NOT_AUTHORIZED	5	Not authorized to perform the requested operation.
REASON_CODE_SERVER_CONNECT_ERROR	32103	Unable to connect to server.
REASON_CODE_SOCKET_FACTORY_MISMATCH	32105	Server URI and supplied SocketFactory do not match.
REASON_CODE_SSL_CONFIG_ERROR	32106	SSL configuration error.
REASON_CODE_UNEXPECTED_ERROR	6	An unexpected error has occurred.

Tracing the telemetry (MQXR) service

Follow these instructions to start a trace of the telemetry service, set the parameters that control the trace, and find the trace output.

Before you begin

Tracing is a support function. Follow these instructions if an IBM service engineer asks you to trace your telemetry (MQXR) service. The product documentation does not document the format of the trace file, or how to use it to debug a client.

About this task

You can use the IBM MQ **strmqtrc** and **endmqtrc** commands to start and stop IBM MQ trace. **strmqtrc** captures trace for the telemetry (MQXR) service. When using **strmqtrc**, there is a delay of up to a couple of seconds before the telemetry service trace is started. For further information about IBM MQ trace, see Tracing. Alternatively, you can trace the telemetry service by using the following procedure:

Procedure

1. Set the trace options to control the amount of detail and the size of the trace. The options apply to a trace started with either the **strmqtrc** or the **controlMQXRChannel** command.

   Set the trace options in the following files:

   `mqxrtrace.properties`

   `trace.config`

   The files are in the following directory:
   - On Windows systems: *`WebSphere MQ data directory`*`\qmgrs\`*`qMgrName`*`\mqxr`.
   - On AIX or Linux systems: `var/mqm/qmgrs/`*`qMgrName`*`/mqxr`.
2. Open a command window in the following directory:
   - On Windows systems: *`WebSphere MQ installation directory`*`\mqxr\bin`.
   - On AIX or Linux systems: `/opt/mqm/mqxr/bin`.
3. Run the following command to start an `SYSTEM.MQXR.SERVICE` trace:

```
►►─┬─./controlMQXRChannel.sh─┬──-qmgr=──qMgrName──-mode=──┬─starttrace─┬────────────────────►
   └─controlMQXRChannel.bat──┘                            └─stoptrace──┘

►─┬─────────────────────────────┬──────────────────────────────────────────────────────────►◄
  └─-clientid=──ClientIdentifier─┘
```

Mandatory parameters

`qmgr=` *`qmgrName`*
: Set *qmgrName* to the queue manager name

`mode= starttrace | stoptrace`
: Set starttrace to begin tracing or to stoptrace to end tracing

Optional parameters

`clientid=` *`ClientIdentifier`*
: Set *ClientIdentifier* to the `ClientIdentifier` of a client. clientid filters trace to a single client. Run the trace command multiple times to trace multiple clients.

For example:

```
/opt/mqm/mqxr/bin/controlMQXRChannel.sh -qmgr=QM1 -mode=starttrace -clientid= problemclient
```

Results

To view the trace output, go to the following directory:

- On Windows systems: `WebSphere MQ data directory\trace`.
- On AIX or Linux systems: `/var/mqm/trace`.

Trace files are named `mqxr_PPPPP.trc`, where `PPPPP` is the process ID.

Related information:

strmqtrc

Tracing the MQTT v3 Java client

Follow these instructions to create an MQTT Java client trace and control its output.

Before you begin

Tracing is a support function. Follow these instructions if an IBM service engineer asks you to trace your MQTT Java client. The product documentation does not document the format of the trace file, or how to use it to debug a client.

Trace only works for the IBM MQ Telemetry Java client.

About this task

Note: The examples are coded for Windows. Change the syntax to run the examples on Linux[7] .

Procedure

1. Create a Java properties file containing the trace configuration.

   In the properties file specify the following optional properties. If a property key is specified more than once, the last occurrence sets the property.

   a. `com.ibm.micro.client.mqttv3.trace.outputName`

      The directory to write the trace file to. It defaults to the client working directory. The trace file is called `mqtt-` *`n`*`.trc`.

      ```
      com.ibm.micro.client.mqttv3.trace.outputName=c:\\MQTT_Trace
      ```

   b. `com.ibm.micro.client.mqttv3.trace.count`

      The number of trace files to write. The default is one file, of unlimited size.

      ```
      com.ibm.micro.client.mqttv3.trace.count=5
      ```

   c. `com.ibm.micro.client.mqttv3.trace.limit`

      The maximum size of file to write, the default is 500000. The limit only applies if more than one trace file is requested.

      ```
      com.ibm.micro.client.mqttv3.trace.limit=100000
      ```

   d. `com.ibm.micro.client.mqttv3.trace.client.` *`clientIdentifier`*`.status`

      Turn trace on or off, per client. If *`clientIdentifier`* `=*`, trace is turned on or off for all clients. By default, trace is turned off for all clients.

      ```
      com.ibm.micro.client.mqttv3.trace.client.*.status=on
      com.ibm.micro.client.mqttv3.trace.client.Client10.status=on
      ```

2. Pass the trace properties file to the JVM using a system property.

   ```
   -Dcom.ibm.micro.client.mqttv3.trace=c:\\MqttTrace.properties
   ```

3. Run the client.

7. Java uses the correct path delimiter. You can code the delimiter in a property file as '/' or '\\' ; '\' is the escape character

4. Convert the trace file from binary encoding to text or `.html`. Use the following command:

   ```
   com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter [-i traceFile] [-o outputFile] [-h] [-d
   time]
   ```

   where the arguments are:

   -? Displays help

   -i traceFile
   Required. Passes in the input file (for example, `mqtt-0.trc`).

   -o outputFile
   Required. Defines the output file (for example, `mqtt-0.trc.html` or `mqtt-0.trc.txt`).

   -h Output as HTML. The output files extension must be `.html`. If not specified, the output is plain text.

   -d time
   Indents a line with `*` if the time difference in milliseconds is greater than or equal to (>=) time. Not applicable for HTML output.

   The following example will output the trace file in HTML format

   ```
   com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o mqtt-0.trc.html -h
   ```

   The second example will output the trace file as plain text, with any consecutive timestamps that have milliseconds with a difference of 50 or greater indented with an asterisk (`*`).

   ```
   com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o mqtt-0.trc.txt -d 50
   ```

   The final example will output the trace file as plain text:

   ```
   com.ibm.micro.client.mqttv3.internal.trace.TraceFormatter -i mqtt-0.trc -o mqtt-0.trc.txt
   ```

System requirements for using SHA-2 cipher suites with MQTT channels

If you use a version of Java that supports SHA-2 cipher suites, you can use these suites to secure your MQTT (telemetry) channels and client apps.

For IBM MQ Version 8.0 , which includes the telemetry (MQXR) service, the minimum Java version is Java 7 from IBM , SR6. SHA-2 cipher suites are supported by default in Java 7 from IBM, SR4 onwards. You can therefore use SHA-2 cipher suites with the telemetry (MQXR) service to secure your MQTT (telemetry) channels.

If you are running an MQTT client with a different JRE, you need to ensure that it also supports the SHA-2 cipher suites. Details of the SHA-2 cipher suite support for each client is given in System requirements for using SHA-2 cipher suites with MQTT clients.

Related information:
Telemetry (MQXR) service
Telemetry channel configuration for MQTT client authentication using SSL
Telemetry channel configuration for channel authentication using SSL
DEFINE CHANNEL (MQTT)
ALTER CHANNEL (MQTT)

Resolving problem: MQTT client does not connect

Resolve the problem of an MQTT client program failing to connect to the telemetry (MQXR) service.

Before you begin

Is the problem at the server, at the client, or with the connection? Have you have written your own MQTT v3 protocol handling client, or an MQTT client application using the C or Java IBM MQTT clients?

Run the verification application supplied with IBM MQ Telemetry on the server, and check that the telemetry channel and telemetry (MQXR) service are running correctly. Then transfer the verification application to the client, and run the verification application there.

About this task

There are a number of reasons why an MQTT client might not connect, or you might conclude it has not connected, to the telemetry server.

Procedure

1. Consider what inferences can be drawn from the reason code that the telemetry (MQXR) service returned to MqttClient.Connect. What type of connection failure is it?

Option	**Description**
`REASON_CODE_INVALID_PROTOCOL_VERSION`	Make sure that the socket address corresponds to a telemetry channel, and you have not used the same socket address for another broker.
`REASON_CODE_INVALID_CLIENT_ID`	Check that the client identifier is no longer than 23 bytes, and contains only characters from the range: `A-Z, a-z, 0-9, './_%`
`REASON_CODE_SERVER_CONNECT_ERROR`	Check that the telemetry (MQXR) service and the queue manager are running normally. Use **`netstat`** to check that the socket address is not allocated to another application.

   If you have written an MQTT client library rather than use one of the libraries provided by IBM MQ Telemetry, look at the `CONNACK` return code.

   From these three errors you can infer that the client has connected to the telemetry (MQXR) service, but the service has found an error.

2. Consider what inferences can be drawn from the reason codes that the client produces when the telemetry (MQXR) service does not respond:

Option	**Description**
`REASON_CODE_CLIENT_EXCEPTION` `REASON_CODE_CLIENT_TIMEOUT`	Look for an FDC file at the server; see "Server-side logs" on page 1829. When the telemetry (MQXR) service detects the client has timed out, it writes a first-failure data capture (FDC) file. It writes an FDC file whenever the connection is unexpectedly broken.

The telemetry (MQXR) service might not have responded to the client, and the timeout at the client expires. The IBM MQ Telemetry Java client only hangs if the application has set an indefinite timeout. The client throws one of these exceptions after the timeout set for MqttClient.Connect expires with an undiagnosed connection problem.

Unless you find an FDC file that correlates with the connection failure you cannot infer that the client tried to connect to the server:

a. Confirm that the client sent a connection request.

   Check the TCPIP request with a tool such as **`tcpmon`**, available from (for example) http://code.google.com/p/tcpmon/

b. Does the remote socket address used by the client match the socket address defined for the telemetry channel?

   The default file persistence class in the Java SE MQTT client supplied with IBM WebSphere MQ Telemetry creates a folder with the name: *clientIdentifier*-**tcp***hostNameport* or *clientIdentifier*-**ssl***hostNameport* in the client working directory. The folder name tells you the `hostName` and `port` used in the connection attempt ; see "Client-side log files" on page 1830.

c. Can you ping the remote server address?

d. Does **`netstat`** on the server show the telemetry channel is running on the port the client is connecting too?

3. Check whether the telemetry (MQXR) service found a problem in the client request.

   The telemetry (MQXR) service writes errors it detects into `mqxr.log`, and the queue manager writes errors into `AMQERR01.LOG` ; see

4. Attempt to isolate the problem by running another client.

   - Run the MQTT sample application using the same telemetry channel.
   - Run the **`wmqttSample`** GUI client to verify the connection. Get **`wmqttSample`** by downloading SupportPac IA92.

     Note: Older versions of IA92 do not include the MQTT v3 Java client library.

   Run the sample programs on the server platform to eliminate uncertainties about the network connection, then run the samples on the client platform.

5. Other things to check:

   a. Are tens of thousands of MQTT clients trying to connect at the same time?

      Telemetry channels have a queue to buffer a backlog of incoming connections. Connections are processed in excess of 10,000 a second. The size of the backlog buffer is configurable using the telemetry channel wizard in IBM MQ Explorer. Its default size is 4096. Check that the backlog has not been configured to a low value.

   b. Are the telemetry (MQXR) service and queue manager still running?

   c. Has the client connected to a high availability queue manager that has switched its TCPIP address?

   d. Is a firewall selectively filtering outbound or return data packets?

Resolving problem: MQTT client connection dropped

Find out what is causing a client to throw unexpected ConnectionLost exceptions after successfully connecting and running for either a short or long while.

Before you begin

The MQTT client has connected successfully. The client might be up for a long while. If clients are starting with only a short interval between them, the time between connecting successfully and the connection being dropped might be short.

It is not hard to distinguish a dropped connection from a connection that was successfully made, and then later dropped. A dropped connection is defined by the MQTT client calling the MqttCallback.ConnectionLost method. The method is only called after the connection has been successfully established. The symptom is different to MqttClient.Connect throwing an exception after receiving a negative acknowledgment or timing out.

If the MQTT client application is not using the MQTT client libraries supplied by IBM MQ, the symptom depends on the client. In the MQTT v3 protocol, the symptom is a lack of timely response to a request to the server, or the failure of the TCP/IP connection.

About this task

The MQTT client calls MqttCallback.ConnectionLost with a throwable exception in response to any server-side problems encountered after receiving a positive connection acknowledgment. When an MQTT client returns from MqttTopic.publish and MqttClient.subscribe the request is transferred to an MQTT client thread that is responsible for sending and receiving messages. Server-side errors are reported asynchronously by passing a throwable exception to the ConnectionLost callback method.

The telemetry (MQXR) service always writes a first-failure data capture file if it drops the connection.

Procedure

1. Has another client started that used the same `ClientIdentifier` ?

   If a second client is started, or the same client is restarted, using the same `ClientIdentifier`, the first connection to the first client is dropped.
2. Has the client accessed a topic that it is not authorized to publish or subscribe to?

   Any actions the telemetry service takes on behalf of a client that return `MQCC_FAIL` result in the service dropping the client connection.

   The reason code is not returned to the client.
   - Look for log messages in the `mqxr.log` and `AMQERR01.LOG` files for the queue manager the client is connected to; see "Server-side logs" on page 1829.
3. Has the TCP/IP connection dropped?

   A firewall might have a low timeout setting for marking a TCPIP connection as inactive, and dropped the connection.
   - Shorten the inactive TCPIP connection time using `MqttConnectOptions.setKeepAliveInterval`.

Resolving problem: Lost messages in an MQTT application

Resolve the problem of losing a message. Is the message non-persistent, sent to the wrong place, or never sent? A wrongly coded client program might lose messages.

Before you begin

How certain are you that the message you sent, was lost? Can you infer that a message is lost because the message was not received? If message is a publication, which message is lost: the message sent by the publisher, or the message sent to the subscriber? Or did the subscription get lost, and the broker is not sending publications for that subscription to the subscriber?

If the solution involves distributed publish/subscribe, using clusters or publish/subscribe hierarchies, there are numerous configuration issues that might result in the appearance of a lost message.

If you sent a message with "At least once" or "At most once" quality of service, it is likely that the message you think is lost was not delivered in the way you expected. It is unlikely that the message has been wrongly deleted from the system. It might have failed to create the publication or the subscription you expected.

The most important step you take in doing problem determination of lost messages is to confirm the message is lost. Recreate the scenario and lose more messages. Use the "At least once" or "At most once" quality of service to eliminate all cases of the system discarding messages.

About this task

There are four legs to diagnosing a lost message.

1. "Fire and forget" messages working as-designed. "Fire and forget" messages are sometimes discarded by the system.
2. Configuration: setting up publish/subscribe with the correct authorities in a distributed environment is not straightforward.
3. Client programming errors: the responsibility for message delivery is not solely the responsibility of code written by IBM.
4. If you have exhausted all these possibilities, you might decide to involve IBM service.

Procedure

1. If the lost message had the "Fire and forget" quality of service, set the "At least once" or "At most once" quality of service. Attempt to lose the message again.
   - Messages sent with "Fire and forget" quality of service are thrown away by IBM MQ in a number of circumstances:
     - Communications loss and channel stopped.
     - Queue manager shut down.
     - Excessive number of messages.
   - The delivery of "Fire and forget" messages depends upon the reliability of TCP/IP. TCP/IP continues to send data packets again until their delivery is acknowledged. If the TCP/IP session is broken, messages with the "Fire and forget" quality of service are lost. The session might be broken by the client or server closing down, a communications problem, or a firewall disconnecting the session.
2. Check that client is restarting the previous session, in order to send undelivered messages with "At least once" or "At most once" quality of service again.
   a. If the client application is using the Java SE MQTT client, check that it sets MqttClient.CleanSession to `false`
   b. If you are using different client libraries, check that a session is being restarted correctly.

3. Check that the client application is restarting the same session, and not starting a different session by mistake.

   To start the same session again, `cleanSession = false`, and the `Mqttclient.clientIdentifier` and the `MqttClient.serverURI` must be the same as the previous session.
4. If a session closes prematurely, check that the message is available in the persistence store at the client to send again.
   a. If the client application is using the Java SE MQTT client, check that the message is being saved in the persistence folder; see "Client-side log files" on page 1830
   b. If you are using different client libraries, or you have implemented your own persistence mechanism, check that it is working correctly.
5. Check that no one has deleted the message before it was delivered.

   Undelivered messages awaiting delivery to MQTT clients are stored in `SYSTEM.MQTT.TRANSMIT.QUEUE`. Messages awaiting delivery to the telemetry server are stored by the client persistence mechanism; see Message persistence in MQTT clients.
6. Check that the client has a subscription for the publication it expects to receive.

   List subscriptions using MQ Explorer, or by using **runmqsc** or PCF commands. All MQTT client subscriptions are named. They are given a name of the form: *ClientIdentifier*:*Topic name*
7. Check that the publisher has authority to publish, and the subscriber to subscribe to the publication topic.

   ```
   dspmqaut -m qMgr -n topicName -t topic -p user ID
   ```

   In a clustered publish/subscribe system, the subscriber must be authorized to the topic on the queue manager to which the subscriber is connected. It is not necessary for the subscriber to be authorized to subscribe to the topic on the queue manager where the publication is published. The channels between the queue managers must be correctly authorized to pass on the proxy subscription and forward the publication.

   Create the same subscription and publish to it using IBM MQ Explorer. Simulate your application client publishing and subscribing by using the client utility. Start the utility from IBM MQ Explorer and change its user ID to match the one adopted by your client application.
8. Check that the subscriber has permission to put the publication on the `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```
9. Check that the IBM MQ point-to-point application has authority to put its message on the `SYSTEM.MQTT.TRANSMIT.QUEUE`.

   ```
   dspmqaut -m qMgr -n queueName -t queue -p user ID
   ```

   See Sending a message to a client directly.

Resolving problem: Telemetry (MQXR) service does not start

Resolve the problem of the telemetry (MQXR) service failing to start. Check the IBM MQ Telemetry installation and no files are missing, moved, or have the wrong permissions. Check the paths used by the telemetry (MQXR) service locate the telemetry (MQXR) service programs.

Before you begin

The WebSphere MQ Telemetry feature is installed. The IBM MQ Explorer has a Telemetry folder in **IBM MQ > Queue Managers >** ***qMgrName*** **> Telemetry**. If the folder does not exist, the installation has failed.

The Telemetry (MQXR) service must have been created for it to start. If the telemetry (MQXR) service has not been created, then run the **Define sample configuration...** wizard in the `Telemetry` folder.

If the telemetry (MQXR) service has been started before, then additional **Channels** and **Channel Status** folders are created under the `Telemetry` folder. The Telemetry service, `SYSTEM.MQXR.SERVICE`, is in the **Services** folder. It is visible if the Explorer radio button to show System Objects is clicked.

Right click `SYSTEM.MQXR.SERVICE` to start and stop the service, show its status, and display whether your user ID has authority to start the service.

About this task

The `SYSTEM.MQXR.SERVICE` telemetry (MQXR) service fails to start. A failure to start manifests itself in two different ways:

1. The start command fails immediately.
2. The start command succeeds, and is immediately followed by the service stopping.

Procedure

1. Start the service

   Result The service stops immediately. A window displays an error message; for example:

   ```
   IBM MQ cannot process the request because the
   executable specified cannot be started. (AMQ4160)
   ```

   Reason

   Files are missing from the installation, or the permissions on installed files are set wrongly.

   The IBM MQ Telemetry feature is installed only on one of a pair of highly available queue managers. If the queue manager instance switches over to a standby, it tries to start `SYSTEM.MQXR.SERVICE`. The command to start the service fails because the telemetry (MQXR) service is not installed on the standby.

   Investigation

   Look in error logs; see "Server-side logs" on page 1829.

   Actions

   Install, or uninstall and reinstall the IBM MQ Telemetry feature.

2. Start the service; wait for 30 seconds; refresh the Explorer and check the service status.

   Result The service starts and then stops.

   Reason

   `SYSTEM.MQXR.SERVICE` started the **`runMQXRService`** command, but the command failed.

   Investigation

   Look in error logs; see "Server-side logs" on page 1829.

See if the problem occurs with only the sample channel defined. Backup and the clear the contents of the *WMQ data directory*\Qmgrs*qMgrName*\mqxr\ directory. Run the sample configuration wizard and try to start the service.

Actions
Look for permission and path problems.

Resolving problem: JAAS login module not called by the telemetry service

Find out if your JAAS login module is not being called by the telemetry (MQXR) service, and configure JAAS to correct the problem.

Before you begin

You have modified *WMQ installation directory*\mqxr\samples\LoginModule.java to create your own authentication class *WMQ installation directory*\mqxr\samples\samples\LoginModule.class. Alternatively, you have written your own JAAS authentication classes and placed them in a directory of your choosing. After some initial testing with the telemetry (MQXR) service, you suspect that your authentication class is not being called by the telemetry (MQXR) service.

Note: Guard against the possibility that your authentication classes might be overwritten by maintenance being applied to IBM MQ. Use your own path for authentication classes, rather than a path within the IBM MQ directory tree.

About this task

The task uses a scenario to illustrate how to resolve the problem. In the scenario, a package called security.jaas contains a JAAS authentication class called JAASLogin.class. It is stored in the path C:\WMQTelemetryApps\security\jaas. Refer to Telemetry channel JAAS configuration for help in configuring JAAS for IBM MQ Telemetry. The example, "Example JAAS configuration" on page 1842 is a sample configuration.

Procedure

1. Look in mqxr.log for an exception thrown by javax.security.auth.login.LoginException.

   See "Server-side logs" on page 1829 for the path to mqxr.log, and Figure 147 on page 1844 for an example of the exception listed in the log.
2. Correct your JAAS configuration by comparing it with the worked example in "Example JAAS configuration" on page 1842.
3. Replace your login class by the sample JAASLoginModule, after refactoring it into your authentication package and deploy it using the same path. Switch the value of loggedIn between true and false.

   If the problem goes away when loggedIn is true, and appears the same when loggedIn is false, the problem lies in your login class.
4. Check whether the problem is with authorization rather than authentication.
   a. Change the telemetry channel definition to perform authorization checking using a fixed user ID. Select a user ID that is a member of the mqm group.
   b. Rerun the client application.

      If the problem disappears, the solution lies with the user ID being passed for authorization. What is the user name being passed? Print it to file from your login module. Check its access permissions using IBM MQ Explorer, or **dspmqauth**.

Example JAAS configuration

Use the **New telemetry channel** wizard, in IBM MQ Explorer, to configure a telemetry channel. The client connects on port 1884, and connects to the `JAASMCAUser` telemetry channel. Figure 141 shows an example of the telemetry properties file created by the telemetry wizard. Do not edit this file directly. The channel authenticates using JAAS, using the configuration called `JAASConfig`. Once the client has authenticated, it uses the user ID `Admin` to authorize its access to IBM MQ objects.

```
com.ibm.mq.MQXR.channel/JAASMCAUser: \
com.ibm.mq.MQXR.Port=1884;\
com.ibm.mq.MQXR.JAASConfig=JAASConfig;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

Figure 141. `WMQ Installation directory\data\qmgrs\qMgrName\mqxr\mqxr_win.properties`

The JAAS configuration file has a stanza named `JAASConfig` that names the Java class `security.jaas.JAASLogin`, which `JAAS` is to use to authenticate clients.

```
JAASConfig {
  security.jaas.JAASLogin required debug=true;
};
```

Figure 142. `WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config`

When `SYSTEM.MQTT.SERVICE` starts, it adds the path in Figure 143 to its classpath.

```
CLASSPATH=C:\WMQTelemtryApps;
```

Figure 143. `WMQ Installation directory\data\qmgrs\qMgrName\service.env`

Figure 144 shows the additional path in Figure 143 added to the classpath that is set up for the telemetry (MQXR) service.

```
CLASSPATH=;C:\IBM\MQ\Program\mqxr\bin\\..\lib\MQXRListener.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\WMQCommonServices.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\objectManager.utils.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\lib\com.ibm.micro.xr.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mq.jmqi.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mqjms.jar;
C:\IBM\MQ\Program\mqxr\bin\\..\..\java\lib\com.ibm.mq.jar;
C:\WMQTelemtryApps;
```

Figure 144. Classpath output from runMQXRService.bat

The output in Figure 145 on page 1843 shows that the telemetry (MQXR) service has started with the channel definition shown in Figure 141.

```
21/05/2010 15:32:12 [main] com.ibm.mq.MQXRService.MQXRPropertiesFile
AMQXR2011I: Property com.ibm.mq.MQXR.channel/JAASMCAUser value
com.ibm.mq.MQXR.Port=1884;
com.ibm.mq.MQXR.JAASConfig=JAASConfig;
com.ibm.mq.MQXR.UserName=Admin;
com.ibm.mq.MQXR.StartWithMQXRService=true
```

Figure 145. WMQ Installation directory\data\qmgrs\qMgrName\errors\ mqxr.log

When the client application connects to the JAAS channel, if `com.ibm.mq.MQXR.JAASConfig=JAASWrongConfig` does not match the name of a JAAS stanza in the `jaas.config` file, the connection fails, and the client throws an exception with a return code of `0` ; see Figure 146. The second exception, `Client is not connected (32104)`, was thrown because the client attempted to disconnect when it was not connected.

```
C:\WMQTelemetryApps>java com.ibm.mq.id.PubAsyncRestartable
Starting a clean session for instance "Admin_PubAsyncRestartab"
Publishing "Hello World Fri May 21 17:23:23 BST 2010" on topic "MQTT Example"
for client instance: "Admin_PubAsyncRestartab" using QoS=1 on address tcp://localhost:1884"
Userid: "Admin", Password: "Password"
Delivery token "528752516" has been received: false
Connection lost on instance "Admin_PubAsyncRestartab" with cause "MqttException"
MqttException (0) - java.io.EOFException
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:118)
        at java.lang.Thread.run(Thread.java:801)
Caused by: java.io.EOFException
        at java.io.DataInputStream.readByte(DataInputStream.java:269)
        at com.ibm.micro.client.mqttv3.internal.wire.MqttInputStream.readMqttWireMessage(MqttInputStream.java:56)
        at com.ibm.micro.client.mqttv3.internal.CommsReceiver.run(CommsReceiver.java:90)
        ... 1 more
Client is not connected (32104)
        at com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:100)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:117)
        at com.ibm.micro.client.mqttv3.internal.ClientComms.disconnect(ClientComms.java:229)
        at com.ibm.micro.client.mqttv3.MqttClient.disconnect(MqttClient.java:385)
        at com.ibm.mq.id.PubAsyncRestartable.main(PubAsyncRestartable.java:49)
```

Figure 146. Exception thrown connecting com.ibm.mq.id.PubAsyncRestartable

`mqxr.log` contains additional output shown in Figure 146.

The error is detected by JAAS which throws `javax.security.auth.login.LoginException` with the cause `No LoginModules configured for JAAS`. It could be caused, as in Figure 147 on page 1844, by a bad configuration name. It might also be the result of other problems JAAS has encountered loading the JAAS configuration.

If no exception is reported by JAAS, JAAS has successfully loaded the `security.jaas.JAASLogin` class named in the `JAASConfig` stanza.

```
21/05/2010 12:06:12 [ServerWorker0] com.ibm.mq.MQXRService.MQTTCommunications
AMQXR2050E: Unable to load JAAS config: JAASWrongConfig.
The following exception occurred javax.security.auth.login.LoginException:
No LoginModules configured for JAAS
```

Figure 147. `mqxr.log` - error loading JAAS configuration

Resolving problem: Starting or running the daemon

Consult the IBM MQ MQTT daemon for devices console log, turn on tracing, or use the symptom table in this topic to troubleshoot problems with the daemon.

Before you begin

Note: The daemon for devices is no longer available. For an alternative solution, see the eclipse.org "Mosquitto" project (https://eclipse.org/mosquitto).

Procedure

1. Check the console log.

   If the daemon is running in the foreground, the console messages are written to the terminal window. If the daemon has been started in the background, the console is where you have redirected `stdout` to.
2. Restart the daemon.

   Changes to the configuration file are not activated until the daemon is restarted.
3. Consult Table 175:

Table 175. Symptom table

Problem	**Suggested solution**

The following message is displayed when you start the daemon on Windows:
 	
`The system cannot execute the specified program`
or
`The application has failed to start because its side-by-side configuration is incorrect.`
	Install Microsoft Visual C++ 2008 Redistributable Package.

Two or more daemons or MQTT-capable servers are inter-connected by a bridge or bridges, and the processor is showing excessive load.	There is possibly a message loop, with one or more messages being repeatedly passed from one server to another. Examine the topic parameters in the configuration files. Use more specific topics where possible. Broad wildcard characters in both directions are the most common cause of connection loops.
The bridge is unable to connect to a remote MQTT-capable server that other MQTT clients can connect to.	The remote server might be incompatible with attempts to determine if the remote server is also an MQTT daemon for devices. Try setting **`try_private`** to `off` to disable special processing to eliminate message loops.
This message is printed when a bridge is configured: `Warning: Connect was not first packet on socket 1888, got CONNACK.`	You have probably configured a bridge to loop back to the local daemon. Loopback is not supported.

Resolving problem: MQTT clients not connecting to the daemon

Clients are not connecting to the daemon, or the daemon is not connecting to other daemons or to an IBM MQ telemetry channel.

Before you begin

Note: The daemon for devices is no longer available. For an alternative solution, see the eclipse.org "Mosquitto" project (https://eclipse.org/mosquitto).

About this task

Trace each MQTT packet sent and received by the daemon.

Procedure

Set the **`trace_output`** parameter to `protocol` in the daemon configuration file or send a command to the daemon.
See Transfer messages between the IBM MQ MQTT daemon for devices and IBM MQ.
Using the protocol setting, the daemon prints a message to the console describing each MQTT packet it sends and receives.

Recovering after failure

Follow a set of procedures to recover after a serious problem.

About this task

Use the recovery methods described here if you cannot resolve the underlying problem by using the diagnostic techniques described throughout the Troubleshooting and support section. If your problem cannot be resolved by using these recovery techniques, contact your IBM Support Center.

Procedure

See the following links for instructions on how to recover from different types of failures:
- "Disk drive failures" on page 1846
- "Damaged queue manager object" on page 1847
- "Damaged single object" on page 1847
- "Automatic media recovery failure" on page 1847

z/OS

See the following links for instructions on how to recover from different types of failures on IBM MQ for z/OS:

- z/OS "Shared queue problems" on page 1849
- z/OS "Active log problems" on page 1849
- z/OS "Archive log problems" on page 1855
- z/OS "BSDS problems" on page 1857
- z/OS "Page set problems" on page 1864
- z/OS "Coupling facility and Db2 problems" on page 1865
- z/OS "Problems with long-running units of work" on page 1868
- z/OS "IMS-related problems" on page 1869

- z/OS "Hardware problems" on page 1871

Related concepts:

"Troubleshooting and support" on page 1275
If you are having problems with your queue manager network or IBM MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 1276
Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

"Making initial checks on Windows, UNIX and Linux systems" on page 1277
Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

Related tasks:

"Contacting IBM Software Support" on page 1658
Grade the severity of the problem, describe the problem and gather background information, then report the problem to IBM Software Support.

Related information:

Backing up and restoring IBM MQ

z/OS Planning for backup and recovery on z/OS

Disk drive failures

You might have problems with a disk drive containing either the queue manager data, the log, or both. Problems can include data loss or corruption. The three cases differ only in the part of the data that survives, if any.

In ***all*** cases first check the directory structure for any damage and, if necessary, repair such damage. If you lose queue manager data, the queue manager directory structure might have been damaged. If so, re-create the directory tree manually before you restart the queue manager.

If damage has occurred to the queue manager data files, but not to the queue manager log files, then the queue manager will normally be able to restart. If any damage has occurred to the queue manager log files, then it is likely that the queue manager will not be able to restart.

Having checked for structural damage, there are a number of things you can do, depending on the type of logging that you use.

- **Where there is major damage to the directory structure or any damage to the log**, remove all the old files back to the `QMgrName` level, including the configuration files, the log, and the queue manager directory, restore the last backup, and restart the queue manager.
- **For linear logging with media recovery**, ensure that the directory structure is intact and restart the queue manager. If the queue manager restarts, check, using MQSC commands such as DISPLAY QUEUE, whether any other objects have been damaged. Recover those you find, using the **rcrmqobj** command. For example:

  ```
  rcrmqobj -m QMgrName -t all *
  ```

  where `QMgrName` is the queue manager being recovered. `-t all *` indicates that all damaged objects of any type are to be recovered. If only one or two objects have been reported as damaged, you can specify those objects by name and type here.
- **For linear logging with media recovery and with an undamaged log**, you might be able to restore a backup of the queue manager data leaving the existing log files and log control file unchanged. Starting the queue manager applies the changes from the log to bring the queue manager back to its state when the failure occurred.

This method relies on two things:

1. You must restore the checkpoint file as part of the queue manager data. This file contains the information determining how much of the data in the log must be applied to give a consistent queue manager.
2. You must have the oldest log file required to start the queue manager at the time of the backup, and all subsequent log files, available in the log file directory.

If this is not possible, restore a backup of both the queue manager data and the log, both of which were taken at the same time. This causes message integrity to be lost.

- **For circular logging**, if the queue manager log files are damaged, restore the queue manager from the latest backup that you have. Once you have restored the backup, restart the queue manager and check for damaged objects. However, because you do not have media recovery, you must find other ways of re-creating the damaged objects.

  If the queue manager log files are not damaged, the queue manager will normally be able to restart. Following the restart you must identify all damaged objects, then delete and redefine them.

Damaged queue manager object

What to do if the queue manager reports a damaged object during normal operation.

There are two ways of recovering in these circumstances, depending on the type of logging you use:

- **For linear logging**, manually delete the file containing the damaged object and restart the queue manager. (You can use the **dspmqfls** command to determine the real, file-system name of the damaged object.) Media recovery of the damaged object is automatic.
- **For circular logging**, restore the last backup of the queue manager data and log, and restart the queue manager.

  There is a further option if you are using circular logging. For a damaged queue, or other object, delete the object and define the object again. In the case of a queue, this option does not allow you to recover any data on the queue.

  Note: Restoring from backup is likely to be out of date, due to the fact that you must have your queue manager shutdown in order to get a clean backup of the queue files.

Damaged single object

If a single object is reported as damaged during normal operation, for linear logging you can re-create the object from its media image. However, for circular logging you cannot re-create a single object.

Automatic media recovery failure

If a local queue required for queue manager startup with a linear log is damaged, and the automatic media recovery fails, restore the last backup of the queue manager data and log and restart the queue manager.

Example recovery procedures on z/OS

Use this topic as a reference for various recovery procedures.

This topic describes procedures for recovering IBM MQ after various error conditions. These error conditions are grouped in the following categories:

Table 176. Example recovery procedures

Problem category	Problem	Where to look next
Shared queue problems	Conflicting definitions for both private and shared queues.	"Shared queue problems" on page 1849
Active log problems	• Dual logging is lost. • Active log has stopped. • One or both copies of the active log data set are damaged. • Write errors on active log data set. • Active log is becoming full or is full. • Read errors on active log data set.	"Active log problems" on page 1849
Archive log problems	• Insufficient DASD space to complete offloading active log data sets. • Offload task has terminated abnormally. • Archive data set allocation problem. 1 • Read I/O errors on the archive data set during restart.	"Archive log problems" on page 1855
BSDS problems	• Error opening BSDS. • Log content does not correspond with BSDS information. • Both copies of the BSDS are damaged. • Unequal time stamps. • Dual BSDS data sets are out of synchronization. • I/O error on BSDS.	"BSDS problems" on page 1857
Page set problems	• Page set full. • A page set has an I/O error.	"Page set problems" on page 1864
coupling facility and Db2 problems	• Storage medium full. • Db2 system fails. • Db2 data-sharing group fails. • Db2 and the coupling facility fail.	"Coupling facility and Db2 problems" on page 1865
Unit of work problems	A long-running unit of work is encountered.	"Problems with long-running units of work" on page 1868
IMS problems	• An IMS application terminates abnormally. • The IMS adapter cannot connect to IBM MQ. • IMS not operational.	"IMS-related problems" on page 1869
Hardware problems	Media recovery procedures	"Hardware problems" on page 1871

Shared queue problems

Problems occur if IBM MQ discovers that a page set based queue, and a shared queue of the same name are defined.

Symptoms
IBM MQ issues the following message:

```
CSQI063E +CSQ1 QUEUE queue-name IS BOTH PRIVATE AND SHARED
```

During queue manager restart, IBM MQ discovered that a page set based queue and a shared queue of the same name coexist.

System action
Once restart processing has completed, any **MQOPEN** request to that queue name fails, indicating the coexistence problem.

System programmer action
None.

Operator action
Delete one version of the queue to allow processing of that queue name. If there are messages on the queue that must be kept, you can use the MOVE QLOCAL command to move them to the other queue.

Active log problems

Use this topic to resolve different problems with the active logs.

This topic covers the following active log problems:

- "Dual logging is lost"
- "Active log stopped" on page 1850
- "One or both copies of the active log data set are damaged" on page 1850
- "Write I/O errors on an active log data set" on page 1851
- "I/O errors occur while reading the active log" on page 1852
- "Active log is becoming full" on page 1853
- Active log is full

Dual logging is lost

Symptoms
IBM MQ issues the following message:

```
CSQJ004I +CSQ1 ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,
          ENDRBA=...
```

Having completed one active log data set, IBM MQ found that the subsequent (COPY n) data sets were not offloaded or were marked stopped.

System action
IBM MQ continues in single mode until offloading has been completed, then returns to dual mode.

System programmer action
None.

Operator action
Check that the offload process is proceeding and is not waiting for a tape mount. You might need to run the print log map utility to determine the state of all data sets. You might also need to define additional data sets.

Active log stopped

Symptoms
IBM MQ issues the following message:

```
CSQJ030E +CSQ1 RBA RANGE startrba TO endrba NOT AVAILABLE IN ACTIVE
          LOG DATA SETS
```

System action
The active log data sets that contain the RBA range reported in message CSQJ030E are unavailable to IBM MQ. The status of these logs is STOPPED in the BSDS. The queue manager terminates with a dump.

System programmer action
You must resolve this problem before restarting the queue manager. The log RBA range must be available for IBM MQ to be recoverable. An active log that is marked as STOPPED in the BSDS will never be reused or archived and this creates a hole in the log.

Look for messages that indicate why the log data set has stopped, and follow the instructions for those messages.

Modify the BSDS active log inventory to reset the STOPPED status. To do this, follow this procedure after the queue manager has terminated:

1. Use the print log utility (CSQJU004) to obtain a copy of the BSDS log inventory. This shows the status of the log data sets.
2. Use the DELETE function of the change log inventory utility (CSQJU003) to delete the active log data sets that are marked as STOPPED.
3. Use the NEWLOG function of CSQJU003 to add the active logs back into the BSDS inventory. The starting and ending RBA for each active log data set must be specified on the NEWLOG statement. (The correct values to use can be found from the print log utility report obtained in Step 1.)
4. Rerun CSQJU004. The active log data sets that were marked as STOPPED are now shown as NEW and NOT REUSABLE. These active logs will be archived in due course.
5. Restart the queue manager.

Note: If your queue manager is running in dual BSDS mode, you must update both BSDS inventories.

One or both copies of the active log data set are damaged

Symptoms
IBM MQ issues the following messages:

```
CSQJ102E +CSQ1 LOG RBA CONTENT OF LOG DATA SET DSNAME=...,
          STARTRBA=..., ENDRBA=...,
          DOES NOT AGREE WITH BSDS INFORMATION
CSQJ232E +CSQ1 OUTPUT DATA SET CONTROL INITIALIZATION PROCESS FAILED
```

System action
Queue manager startup processing is terminated.

System programmer action
If one copy of the data set is damaged, carry out these steps:

1. Rename the damaged active log data set and define a replacement data set.
2. Copy the undamaged data set to the replacement data set.
3. Use the change log inventory utility to:
   - Remove information relating to the damaged data set from the BSDS.
   - Add information relating to the replacement data set to the BSDS.
4. Restart the queue manager.

If both copies of the active log data sets are damaged, the current page sets are available, **and the queue manager shut down cleanly**, carry out these steps:

1. Rename the damaged active log data sets and define replacement data sets.
2. Use the change log records utility to:
   - Remove information relating to the damaged data set from the BSDS.
   - Add information relating to the replacement data set to the BSDS.
3. Rename the current page sets and define replacement page sets.
4. Use CSQUTIL (FORMAT and RESETPAGE) to format the replacement page sets and copy the renamed page sets to them. The RESETPAGE function also resets the log information in the replacement page sets.

If the queue manager did not shut down cleanly, you must either restore your system from a previous known point of consistency, or perform a cold start (described in Reinitializing a queue manager).

Operator action
None.

Write I/O errors on an active log data set

Symptoms
IBM MQ issues the following message:

```
CSQJ105E +CSQ1 csect-name LOG WRITE ERROR DSNAME=...,
          LOGRBA=..., ERROR STATUS=ccccffss
```

System action
IBM MQ carries out these steps:

1. Marks the log data set that has the error as TRUNCATED in the BSDS.
2. Goes on to the next available data set.
3. If dual active logging is used, truncates the other copy at the same point.

The data in the truncated data set is offloaded later, as usual.

The data set will be reused on the next cycle.

System programmer action

None.

Operator action

If errors on this data set still exist, shut down the queue manager after the next offload process. Then use Access Method Services (AMS) and the change log inventory utility to add a replacement. (For instructions, see Changing the BSDS.)

I/O errors occur while reading the active log

Symptoms

IBM MQ issues the following message:

```
CSQJ106E +CSQ1 LOG READ ERROR DSNAME=..., LOGRBA=...,
         ERROR STATUS=ccccffss
```

System action

This depends on when the error occurred:

- If the error occurs during the offload process, the process tries to read the RBA range from a second copy.
  - If no second copy exists, the active log data set is stopped.
  - If the second copy also has an error, only the original data set that triggered the offload process is stopped. The archive log data set is then terminated, leaving a gap in the archived log RBA range.
  - This message is issued:

```
CSQJ124E +CSQ1 OFFLOAD OF ACTIVE LOG SUSPENDED FROM
         RBA xxxxxx TO RBA xxxxxx DUE TO I/O ERROR
```

  - If the second copy is satisfactory, the first copy is not stopped.
- If the error occurs during recovery, IBM MQ provides data from specific log RBAs requested from another copy or archive. If this is unsuccessful, recovery does not succeed, and the queue manager terminates abnormally.
- If the error occurs during restart, if dual logging is used, IBM MQ continues with the alternative log data set, otherwise the queue manager ends abnormally.

System programmer action

Look for system messages, such as IEC prefixed messages, and try to resolve the problem using the recommended actions for these messages.

If the active log data set has been stopped, it is not used for logging. The data set is not deallocated; it is still used for reading. Even if the data set is not stopped, an active log data set that gives persistent errors should be replaced.

Operator action

None.

Replacing the data set

How you replace the data set depends on whether you are using single or dual active logging.

If you are using dual active logging:

1. Ensure that the data has been saved.

   The data is saved on the other active log and this can be copied to a replacement active log.
2. Stop the queue manager and delete the data set with the error using Access Method Services.

3. Redefine a new log data set using Access Method Services DEFINE so that you can write to it. Use DFDSS or Access Method Services REPRO to copy the good log in to the redefined data set so that you have two consistent, correct logs again.
4. Use the change log inventory utility, CSQJU003, to update the information in the BSDS about the corrupted data set as follows:
   a. Use the DELETE function to remove information about the corrupted data set.
   b. Use the NEWLOG function to name the new data set as the new active log data set and give it the RBA range that was successfully copied.

      You can run the DELETE and NEWLOG functions in the same job step. Put the DELETE statement before NEWLOG statement in the SYSIN input data set.
5. Restart the queue manager.

If you are using single active logging:

1. Ensure that the data has been saved.
2. Stop the queue manager.
3. Determine whether the data set with the error has been offloaded:
   a. Use the CSQJU003 utility to list information about the archive log data sets from the BSDS.
   b. Search the list for a data set with an RBA range that includes the RBA of the corrupted data set.
4. If the corrupted data set has been offloaded, copy its backup in the archive log to a new data set. Then, skip to step 6.
5. If an active log data set is stopped, an RBA is not offloaded. Use DFDSS or Access Method Services REPRO to copy the data from the corrupted data set to a new data set.

   If further I/O errors prevent you from copying the entire data set, a gap occurs in the log.

   Note: Queue manager restart will not be successful if a gap in the log is detected.
6. Use the change log inventory utility, CSQJU003, to update the information in the BSDS about the corrupted data set as follows:
   a. Use the DELETE function to remove information about the corrupted data set.
   b. Use the NEWLOG function to name the new data set as the new active log data set and to give it the RBA range that was successfully copied.

      The DELETE and NEWLOG functions can be run in the same job step. Put the DELETE statement before NEWLOG statement in the SYSIN input data set.
7. Restart the queue manager.

Active log is becoming full

The active log can fill up for several reasons, for example, delays in offloading and excessive logging. If an active log runs out of space, this has serious consequences. When the active log becomes full, the queue manager halts processing until an offload process has been completed. If the offload processing stops when the active log is full, the queue manager can end abnormally. Corrective action is required before the queue manager can be restarted.

Symptoms

Because of the serious implications of an active log becoming full, the queue manager issues the following warning message when the last available active log data set is 5% full:

```
CSQJ110E +CSQ1 LAST COPYn ACTIVE LOG DATA SET IS nnn PERCENT FULL
```

and reissues the message after each additional 5% of the data set space is filled. Each time the message is issued, the offload process is started.

System action
Messages are issued and offload processing started. If the active log becomes full, further actions are taken. See “Active log is full”

System programmer action
Use the DEFINE LOG command to dynamically add further active log data sets. This permits IBM MQ to continue its normal operation while the error causing the offload problems is corrected. For more information about the DEFINE LOG command, see DEFINE LOG.

Active log is full

Symptoms
When the active log becomes full, the queue manager halts processing until an offload process has been completed. If the offload processing stops when the active log is full, the queue manager can end abnormally. Corrective action is required before the queue manager can be restarted.

IBM MQ issues the following message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

and an offload process is started. The queue manager then halts processing until the offload process has been completed.

System action
IBM MQ waits for an available active log data set before resuming normal IBM MQ processing. Normal shut down, with either QUIESCE or FORCE, is not possible because the shutdown sequence requires log space to record system events related to shut down (for example, checkpoint records). If the offload processing stops when the active log is full, the queue manager stops with an X'6C6' abend; restart in this case requires special attention. For more details, see “Problem determination on z/OS” on page 1724.

System programmer action
You can provide additional active log data sets before restarting the queue manager. This permits IBM MQ to continue its normal operation while the error causing the offload process problems is corrected. To add new active log data sets, use the change log inventory utility (CSQJU003) when the queue manager is not active. For more details about adding new active log data sets, see Changing the BSDS.

Consider increasing the number of logs by:

1. Making sure that the queue manager is stopped, then using the Access Method Services DEFINE command to define a new active log data set.
2. Defining the new active log data set in the BSDS using the change log inventory utility (CSQJU003).

When you restart the queue manager, offloading starts automatically during startup, and any work that was in progress when IBM MQ was forced to stop is recovered.

Operator action
Check whether the offload process is waiting for a tape drive. If it is, mount the tape. If you cannot mount the tape, force IBM MQ to stop by using the z/OS CANCEL command.

Archive log problems

Use this topic to investigate, and resolve problems with the archive logs.

This topic covers the following archive log problems:

- "Allocation problems"
- "Offload task terminated abnormally"
- "Insufficient DASD space to complete offload processing" on page 1856
- "Read I/O errors on the archive data set while IBM MQ is restarting" on page 1857

Allocation problems

Symptoms

IBM MQ issues message: CSQJ103E

```
CSQJ103E +CSQ1 LOG ALLOCATION ERROR DSNAME=dsname,
         ERROR STATUS=eeeeiiii, SMS REASON CODE=sss
```

z/OS dynamic allocation provides the ERROR STATUS. If the allocation was for offload processing, the following message is also displayed: CSQJ115E:

```
CSQJ115E +CSQ1 OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE
         DATA SET
```

System action

The following actions take place:

- If the input is needed for recovery, and recovery is not successful, and the queue manager ends abnormally.
- If the active log had become full and an offload task was scheduled but not completed, the offload task tries again the next time it is triggered. The active log does not reuse a data set that has not yet been archived.

System programmer action

None.

Operator action

Check the allocation error code for the cause of the problem, and correct it. Ensure that drives are available, and either restart or wait for the offload task to be retried. Be careful if a DFP/DFSMS ACS user-exit filter has been written for an archive log data set, because this can cause a device allocation error when the queue manager tries to read the archive log data set.

Offload task terminated abnormally

Symptoms

No specific IBM MQ message is issued for write I/O errors.

Only a z/OS error recovery program message appears. If you get IBM MQ message CSQJ128E, the offload task has ended abnormally.

System action

The following actions take place:

- The offload task abandons the output data set; no entry is made in the BSDS.
- The offload task dynamically allocates a new archive and restarts offloading from the point at which it was previously triggered.
- If an error occurs on the new data set:

– In dual archive mode, message CSQJ114I is generated and the offload processing changes to single mode:

```
CSQJ114I +CSQ1 ERROR ON ARCHIVE DATA SET, OFFLOAD
         CONTINUING WITH ONLY ONE ARCHIVE DATA SET BEING
         GENERATED
```

– In single archive mode, the output data set is abandoned. Another attempt to process this RBA range is made the next time offload processing is triggered.

– The active log does not wrap around; if there are no more active logs, data is not lost.

System programmer action
None.

Operator action
Ensure that offload task is allocated on a reliable drive and control unit.

Insufficient DASD space to complete offload processing

Symptoms
While offloading the active log data sets to DASD, the process terminates unexpectedly. IBM MQ issues message CSQJ128E:

```
CSQJ128E +CSQ1 LOG OFF-LOAD TASK FAILED FOR ACTIVE LOG nnnnn
```

The error is preceded by z/OS messages IEC030I, IEC031I, or IEC032I.

System action
IBM MQ de-allocates the data set on which the error occurred. If IBM MQ is running in dual archive mode, IBM MQ changes to single archive mode and continues the offload task. If the offload task cannot be completed in single archive mode, the active log data sets cannot be offloaded, and the state of the active log data sets remains NOT REUSABLE. Another attempt to process the RBA range of the abandoned active log data sets is made the next time the offload task is triggered.

System programmer action
The most likely causes of these symptoms are:

- The size of the archive log data set is too small to contain the data from the active log data sets during offload processing. All the secondary space allocations have been used. This condition is normally accompanied by z/OS message IEC030I. The return code in this message might provide further explanations for the cause of these symptoms.

  To solve the problem

  1. Issue the command `CANCEL <queue-manager name>` to cancel the queue manager job
  2. Increase the primary or secondary allocations (or both) for the archive log data set (in the CSQ6ARVP system parameters), or reduce the size of the active log data set.

     If the data to be offloaded is large, you can mount another online storage volume or make one available to IBM MQ.
  3. Restart the queue manager.

- All available space on the DASD volumes to which the archive data set is being written has been exhausted. This condition is normally accompanied by z/OS message IEC032I.

  To solve the problem, make more space available on the DASD volumes, or make another online storage volume available for IBM MQ.

- The primary space allocation for the archive log data set (as specified in the CSQ6ARVP system parameters) is too large to allocate to any available online DASD device. This condition is normally accompanied by z/OS message IEC032I.

  To solve the problem, make more space available on the DASD volumes, or make another online storage volume available for IBM MQ. If this is not possible, you must adjust the value of PRIQTY in the CSQ6ARVP system parameters to reduce the primary allocation. (For details, see Using CSQ6ARVP.)

  Note: If you reduce the primary allocation, you might have to increase the size of the secondary space allocation to avoid future abends.

Operator action
: None.

Read I/O errors on the archive data set while IBM MQ is restarting

Symptoms
: No specific IBM MQ message is issued; only the z/OS error recovery program message appears.

System action
: This depends on whether a second copy exists:
  - If a second copy exists, it is allocated and used.
  - If a second copy does not exist, restart is not successful.

System programmer action
: None.

Operator action
: Try to restart, using a different drive.

BSDS problems

Use this topic to investigate, and resolve problems with BSDS.

For background information about the bootstrap data set (BSDS), see the Planning your IBM MQ environment on z/OS .

This topic describes the following BSDS problems:
- "Error occurs while opening the BSDS" on page 1858
- "Log content does not agree with the BSDS information" on page 1858
- "Both copies of the BSDS are damaged" on page 1858
- "Unequal time stamps" on page 1859
- "Out of synchronization" on page 1860
- "I/O error" on page 1861
- "Log range problems" on page 1861

Normally, there are two copies of the BSDS, but if one is damaged, IBM MQ immediately changes to single BSDS mode. However, the damaged copy of the BSDS must be recovered before restart. If you are in single mode and damage the only copy of the BSDS, or if you are in dual mode and damage both copies, use the procedure described in Recovering the BSDS.

This section covers some of the BSDS problems that can occur at startup. Problems *not* covered here include:
- RECOVER BSDS command errors (messages CSQJ301E - CSQJ307I)
- Change log inventory utility errors (message CSQJ123E)
- Errors in the BSDS backup being dumped by offload processing (message CSQJ125E)

Error occurs while opening the BSDS

Symptoms

IBM MQ issues the following message:

```
CSQJ100E +CSQ1 ERROR OPENING BSDSn DSNAME=..., ERROR STATUS=eeii
```

where *eeii* is the VSAM return code. For information about VSAM codes, see the *DFSMS/MVS Macro Instructions for Data Sets* documentation.

System action

During system initialization, the startup is terminated.

During a RECOVER BSDS command, the system continues in single BSDS mode.

System programmer action

None.

Operator action

Carry out these steps:

1. Run the print log map utility on both copies of the BSDS, and compare the lists to determine which copy is accurate or current.
2. Rename the data set that had the problem, and define a replacement for it.
3. Copy the accurate data set to the replacement data set, using Access Method Services.
4. Restart the queue manager.

Log content does not agree with the BSDS information

Symptoms

IBM MQ issues the following message:

```
CSQJ102E +CSQ1 LOG RBA CONTENT OF LOG DATA SET DSNAME=...,
          STARTRBA=..., ENDRBA=...,
          DOES NOT AGREE WITH BSDS INFORMATION
```

This message indicates that the change log inventory utility was used incorrectly or that a down-level data set is being used.

System action

Queue manager startup processing is terminated.

System programmer action

None.

Operator action

Run the print log map utility and the change log inventory utility to print and correct the contents of the BSDS.

Both copies of the BSDS are damaged

Symptoms

IBM MQ issues the following messages:

```
CSQJ107E +CSQ1 READ ERROR ON BSDS
         DSNAME=... ERROR STATUS=0874
CSQJ117E +CSQ1 REG8 INITIALIZATION ERROR READING BSDS
         DSNAME=... ERROR STATUS=0874
CSQJ119E +CSQ1 BOOTSTRAP ACCESS INITIALIZATION PROCESSING FAILED
```

System action
Queue manager startup processing is terminated.

System programmer action
Carry out these steps:

1. Rename the data set, and define a replacement for it.
2. Locate the BSDS associated with the most recent archive log data set, and copy it to the replacement data set.
3. Use the print log map utility to print the contents of the replacement BSDS.
4. Use the print log records utility to print a summary report of the active log data sets missing from the replacement BSDS, and to establish the RBA range.
5. Use the change log inventory utility to update the missing active log data set inventory in the replacement BSDS.
6. If dual BSDS data sets had been in use, copy the updated BSDS to the second copy of the BSDS.
7. Restart the queue manager.

Operator action
None.

Unequal time stamps

Symptoms
IBM MQ issues the following message:

```
CSQJ120E +CSQ1 DUAL BSDS DATA SETS HAVE UNEQUAL TIME STAMPS,
         SYSTEM BSDS1=...,BSDS2=...,
         UTILITY BSDS1=...,BSDS2=...
```

The possible causes are:

- One copy of the BSDS has been restored. All information about the restored BSDS is down-level. The down-level BSDS has the lower time stamp.
- One of the volumes containing the BSDS has been restored. All information about the restored volume is down-level. If the volume contains any active log data sets or IBM MQ data, they are also down-level. The down-level volume has the lower time stamp.
- Dual logging has degraded to single logging, and you are trying to start without recovering the damaged log.
- The queue manager terminated abnormally after updating one copy of the BSDS but before updating the second copy.

System action
IBM MQ attempts to resynchronize the BSDS data sets using the more recent copy. If this fails, queue manager startup is terminated.

System programmer action
None.

Operator action

If automatic resynchronization fails, carry out these steps:

1. Run the print log map utility on both copies of the BSDS, compare the lists to determine which copy is accurate or current.
2. Rename the down-level data set and define a replacement for it.
3. Copy the good data set to the replacement data set, using Access Method Services.
4. If applicable, determine whether the volume containing the down-level BSDS has been restored. If it has been restored, all data on that volume, such as the active log data, is also down-level.

   If the restored volume contains active log data and you were using dual active logs on separate volumes, you need to copy the current version of the active log to the down-level log data set. See Recovering logs for details of how to do this.

Out of synchronization

Symptoms

IBM MQ issues the following message during queue manager initialization:

```
CSQJ122E +CSQ1 DUAL BSDS DATA SETS ARE OUT OF SYNCHRONIZATION
```

The system time stamps of the two data sets are identical. Differences can exist if operator errors occurred while the change log inventory utility was being used. (For example, the change log inventory utility was only run on one copy.) The change log inventory utility sets a private time stamp in the BSDS control record when it starts, and a close flag when it ends. IBM MQ checks the change log inventory utility time stamps and, if they are different, or they are the same but one close flag is not set, IBM MQ compares the copies of the BSDSs. If the copies are different, CSQJ122E is issued.

This message is also issued by the BSDS conversion utility if two input BSDS are specified and a record is found that differs between the two BSDS copies. This situation can arise if the queue manager terminated abnormally prior to the BSDS conversion utility being run.

System action

Queue manager startup or the utility is terminated.

System programmer action

None.

Operator action

If the error occurred during queue manager initialization, carry out these steps:

1. Run the print log map utility on both copies of the BSDS, and compare the lists to determine which copy is accurate or current.
2. Rename the data set that had the problem, and define a replacement for it.
3. Copy the accurate data set to the replacement data set, using access method services.
4. Restart the queue manager.

If the error occurred when running the BSDS conversion utility, carry out these steps:

1. Attempt to restart the queue manager and shut it down cleanly before attempting to run the BSDS conversion utility again.
2. If this does not solve the problem, run the print log map utility on both copies of the BSDS, and compare the lists to determine which copy is accurate or current.
3. Change the JCL used to invoke the BSDS conversion utility to specify the current BSDS in the SYSUT1 DD statement, and remove the SYSUT2 DD statement, before submitting the job again.

I/O error

Symptoms
IBM MQ changes to single BSDS mode and issues the user message:

```
CSQJ126E +CSQ1 BSDS ERROR FORCED SINGLE BSDS MODE
```

This is followed by one of the following messages:

```
CSQJ107E +CSQ1 READ ERROR ON BSDS
          DSNAME=... ERROR STATUS=...

CSQJ108E +CSQ1 WRITE ERROR ON BSDS
          DSNAME=... ERROR STATUS=...
```

System action
The BSDS mode changes from dual to single.

System programmer action
None.

Operator action
Carry out these steps:
1. Use Access Method Services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the BSDS that had the error. Example control statements can be found in job CSQ4BREC in thlqual.SCSQPROC.
2. Issue the IBM MQ command RECOVER BSDS to make a copy of the good BSDS in the newly allocated data set and reinstate dual BSDS mode. See also Recovering the BSDS.

Log range problems

Symptoms

IBM MQ has issued message CSQJ113E when reading its own log, or message CSQJ133E or CSQJ134E when reading the log of a queue manager in the queue-sharing group. This can happen when you do not have the archive logs needed to restart the queue manager or recover a CF structure.

System action

Depending upon what log record is being read and why, the requestor might end abnormally with a reason code of `X'00D1032A'`.

System programmer action

Run the print log map utility (CSQJU004) to determine the cause of the error. When message CSQJ133E or CSQJ134E has been issued, run the utility against the BSDS of the queue manager indicated in the message.

If you have:
- Deleted the entry with the log range (containing the log RBA or LRSN indicated in the message) from the BSDS, and
- Not deleted or reused the data set

you can add the entry back into the BSDS using the following procedure:

1. Identify the data set containing the required RBA or LRSN, by looking at an old copy of the contents of BSDS, or by running CSQJU004 against a backup of the BSDS.
2. Add the data set back into the BSDS using the change log inventory utility (CSQJU003).
3. Restart the queue manager.

If an archive log data set has been deleted, you will not be able to recover the page set or CF structure that needs the archive logs. Identify the reason that the queue manager needs to read the log record, then take one of the following actions depending on the page set or CF structure affected.

Page sets

Message CSQJ113E during the recovery phase of queue manager restart indicates that the log is needed to perform media recovery to bring a page set up to date.

Identify the page sets that need the deleted log data set for media recovery, by looking at the media recovery RBA in the CSQI1049I message issued for each page set during queue manager restart, then perform the following actions.

- **Page set zero**

  You can recover the objects on page set zero, by using the following procedure.

  Attention: All data in all other page sets will be lost when you carry out the procedure.

  1. Use function SDEFS of the CSQUTIL utility to produce a file of IBM MQ DEFINE commands.
  2. Format page set zero using CSQUTIL, then redefine the other page sets as described in the next section.
  3. Restart the queue manager.
  4. Use CSQUTIL to redefine the objects using the DEFINE commands produced by the utility in step 1.
- **Page sets 1-99**

  Use the following procedure to redefine the page sets.

  Attention: Any data on the page set is lost when you carry out this operation.

  1. If you can access the page set without any I/O errors, reformat the page set using the CSQUTIL utility with the command FORMAT TYPE(NEW).
  2. If I/O errors occurred when accessing the page set, delete the page set and recreate it.

     If you want the page set to be the same size as before, use the command LISTCAT ENT(*dsname*) ALLOC to obtain the existing space allocations, and use these in the z/OS DEFINE CLUSTER command.

     Format the new page set using the CSQUTIL utility with the command FORMAT TYPE(NEW).
  3. Restart the queue manager. You might have to take certain actions, such as resetting channels or resolving indoubt channels.

CF structures

Messages CSQJ113E, CSQJ133E, or CSQJ134E, during the recovery of a CF structure, indicate that the logs needed to recover the structure are not available on at least one member of the queue-sharing group.

Take one of the following actions depending on the structure affected:

Application CF structure

Issue the command RECOVER CFSTRUCT(*structure-name*) TYPE(PURGE).

This process empties the structure, so any messages on the structure are lost.

CSQSYSAPPL structure
Contact your IBM support center.

Administration structure
This structure is rebuilt using log data since the last checkpoint on each queue manager, which should be in active logs.

If you get this error during administration structure recovery, contact your IBM support center as this indicates that the active log is not available.

Once you have recovered the page set or CF structure, perform a backup of the logs, BSDS, page sets, and CF structures.

To prevent this problem from occurring again, increase the:
- Archive log retention (ARCRETN) value to be longer, and
- Increase the frequency of the CF structure backups.

Recovering a CF structure

Conceptually, the data from the previously backed up CF structure is read from the IBM MQ log; the log is read forwards from the backup and any changes are reapplied to the restored structure.

About this task

The log range to use is found from the latest backup of each structure to be recovered, to the current time. The log range is identified by log range sequence number (LRSN) values.

A LRSN uses the 6 most significant digits of a 'store clock value'.

Note that the whole log (back to the time the structure was created) is read, if you have not done a backup of the structure.

Procedure

1. Check that the logs from each queue manage in the queue-sharing group (QSG) are read for records in this LSRN range. Note that the logs are read backwards.
2. Check that a list of changes for each structure to be recovered is built.
3. Data from the coupling facility (CF) structure backup is read and the data is restored. For example, if the backup was done on queue manager `A`, and the recovery is running on queue manager `B`, queue manager `B` reads the logs from queue manager `A` to restore the structure.

   When the start of the backup of the CF structure is read, an internal task is started to take the restored data for the structure and merge it with the changes read from the log.
4. Check that processing continues for each structure being restored.

Example

In the following example, the command RECOVER CFSTRUCT(APP3) has been issued, and the following messages produced:

```
04:00:00 CSQE132I CDL2 CSQERRPB Structure recovery started, using log range from LRSN=CC56D01026CC
to LRSN=CC56DC368924
This is the start of reading the logs backwards from each qmgr in the QSG from the time
of failure to the to the structure backup. The LRSN values give the ranges being used.
Log records for all structures (just one structure in this example) being recovered are
processed at the same time.

04:02:00 CSQE133I CDL2 CSQERPLS Structure recovery reading log backwards, LRSN=CC56D0414372
This message is produced periodically to show the process

04:02:22 CSQE134I CDL2 CSQERRPB Structure recovery reading log completed
```

```
The above process of replaying the logs backwards has finished,

04:02:22 CSQE130I CDL2 CSQERCF2 Recovery of structure APP3 started, using CDL1 log range
from RBA=000EE86D902E to RBA=000EF5E8E4DC
The task to process the data for APP3 has been started. The last backup of CF structure
APP3 was done on CDL1 within the given RBA range, so this log range has to be read.

04:02:29 CSQE131I CDL2 CSQERCF2 Recovery of structure APP3 completed
The data merge has completed. The structure is recovered.
```

Page set problems

Use this topic to investigate, and resolve problems with the page sets.

This topic covers the problems that you might encounter with page sets:

- "Page set I/O errors" describes what happens if a page set is damaged.
- "Page set full" on page 1865 describes what happens if there is not enough space on the page set for any more MQI operations.

Page set I/O errors

Problem
A page set has an I/O error.

Symptoms
This message is issued:

```
CSQP004E +CSQ1 csect-name I/O ERROR STATUS ret-code
PSID psid RBA rba
```

System action
The queue manager terminates abnormally.

System programmer action
None.

Operator action
Repair the I/O error cause.

If none of the page sets are damaged, restart the queue manager. IBM MQ automatically restores the page set to a consistent state from the logs.

If one or more page sets are damaged:

1. Rename the damaged page sets and define replacement page sets.
2. Copy the most recent backup page sets to the replacement page sets.
3. Restart the queue manager. IBM MQ automatically applies any updates that are necessary from the logs.

You cannot restart the queue manager if page set zero is not available. If one of the other page sets is not available, you can comment out the page set DD statement in the queue manager start-up JCL procedure. This lets you defer recovery of the defective page set, enabling other users to continue accessing IBM MQ.

When you add the page set back to the JCL procedure, system restart reads the log from the point where the page set was removed from the JCL to the end of the log. This procedure could take a long time if numerous data has been logged.

A reason code of MQRC_PAGESET_ERROR is returned to any application that tries to access a queue defined on a page set that is not available.

When you have restored the defective page set, restore its associated DD statement and restart the queue manager.

The operator actions described here are only possible if all log data sets are available. If your log data sets are lost or damaged, see Restarting if you have lost your log data sets.

Page set full

Problem
: There is not enough space on a page set for one of the following:

  - **MQPUT** or **MQPUT1** calls to be completed
  - Object manipulation commands to be completed (for example, DEFINE QLOCAL)
  - **MQOPEN** calls for dynamic queues to be completed

Symptoms
: The request fails with reason code MQRC_STORAGE_MEDIUM_FULL. The queue manager cannot complete the request because there is not enough space remaining on the page set.

  Reason code MQRC_STORAGE_MEDIUM_FULL can occur even when the page set expand attribute is set to EXPAND(USER). Before the reason code MQRC_STORAGE_MEDIUM_FULL is returned to the application code, the queue manager will attempt to expand the page set and retry the API request. On a heavily loaded system it is possible that the expanded storage can be used by other IO operations before the retry of the API. See Managing page sets.

  The cause of this problem could be messages accumulating on a transmission queue because they cannot be sent to another system.

System action
: Further requests that use this page set are blocked until enough messages are removed or objects deleted to make room for the new incoming requests.

Operator action
: Use the IBM MQ command DISPLAY USAGE PSID(*) to identify which page set is full.

System programmer action
: You can either enlarge the page set involved or reduce the loading on that page set by moving queues to another page set. See Managing page sets for more information about these tasks. If the cause of the problem is messages accumulating on the transmission queue, consider starting distributed queuing to transmit the messages.

Coupling facility and Db2 problems

Use this topic to investigate, and resolve problems with the coupling facility, and Db2.

This section covers the problems that you might encounter with the coupling facility and Db2:

- “Storage medium full”
- “A Db2 system fails” on page 1866
- “A Db2 data-sharing group fails” on page 1867
- “Db2 and the coupling facility fail” on page 1867

Storage medium full

Problem
: A coupling facility structure is full.

Symptoms
: If a queue structure becomes full, return code MQRC_STORAGE_MEDIUM_FULL is returned to the application.

If the administration structure becomes full, the exact symptoms depend on which processes experience the error, they might range from no responses to CMDSCOPE(GROUP) commands, to queue manager failure as a result of problems during commit processing.

System programmer action

You can use IBM MQ to inhibit **MQPUT** operations to some of the queues in the structure to prevent applications from writing more messages, start more applications to get messages from the queues, or quiesce some of the applications that are putting messages to the queue.

Alternatively you can use XES facilities to alter the structure size in place. The following z/OS command alters the size of the structure:

```
SETXCF START,ALTER,STRNAME= structure-name,SIZE= newsize
```

where `newsize` is a value that is less than the value of MAXSIZE specified on the CFRM policy for the structure, but greater than the current coupling facility size.

You can monitor the utilization of a coupling facility structure with the DISPLAY CFSTATUS command.

A Db2 system fails

If a Db2 subsystem that IBM MQ is connected to fails, IBM MQ attempts to reconnect to the subsystem, and continue working. If you specified a Db2 group attach name in the QSGDATA parameter of the CSQ6SYSP system parameter module, IBM MQ reconnects to another active Db2 that is a member of the same data-sharing group as the failed Db2, if one is available on the same z/OS image.

There are some queue manager operations that do not work while IBM MQ is not connected to Db2. These are:

- Deleting a shared queue or group object definition.
- Altering, or issuing **MQSET** on, a shared queue or group object definition. The restriction of **MQSET** on shared queues means that operations such as triggering or the generation of performance events do not work correctly.
- Defining new shared queues or group objects.
- Displaying shared queues or group objects.
- Starting, stopping, or other actions for shared channels.
- Reading the shared queue definition from Db2 the first time that the shared queue is open by issuing an MQOPEN.

Other IBM MQ API operations continue to function as normal for shared queues, and all IBM MQ operations can be performed against the queue manager private versions (COPY objects) built from GROUP objects. Similarly, any shared channels that are running continue normally until they end or have an error, when they go into retry state.

When IBM MQ reconnects to Db2, resynchronization is performed between the queue manager and Db2. This involves notifying the queue manager of new objects that have been defined in Db2 while it was disconnected (other queue managers might have been able to continue working as normal on other z/OS images through other Db2 subsystems), and updating object attributes of shared queues that have changed in Db2. Any shared channels in retry state are recovered.

If a Db2 fails, it might have owned locks on Db2 resources at the time of failure. In some cases, this might make certain IBM MQ objects unavailable to other queue managers that are not otherwise affected. To resolve this, restart the failed Db2 so that it can perform recovery processing and release the locks.

A Db2 data-sharing group fails

If an entire Db2 data-sharing group fails, recovery might be to the time of failure, or to a previous point in time.

In the case of recovery to the point of failure, IBM MQ reconnects when Db2 has been recovered, the resynchronization process takes places, and normal queue manager function is resumed.

However, if Db2 is recovered to a previous point in time, there might be inconsistencies between the actual queues in the coupling facility structures and the Db2 view of those queues. For example, at the point in time Db2 is recovered to, a queue existed that has since been deleted and its location in the coupling facility structure reused by the definition of a new queue that now contains messages.

If you find yourself in this situation, you must stop all the queue managers in the queue-sharing group, clear out the coupling facility structures, and restart the queue managers. You must then use IBM MQ commands to define any missing objects. To do this, use the following procedure:

1. Prevent IBM MQ from reconnecting to Db2 by starting Db2 in utility mode, or by altering security profiles.
2. If you have any important messages on shared queues, you might be able to offload them using the COPY function of the CSQUTIL utility program, but this might not work.
3. Terminate all queue managers.
4. Use the following z/OS command to clear all structures:

```
SETXCF FORCE,STRUCTURE,STRNAME=
```

5. Restore Db2 to a historical point in time.
6. Reestablish queue manager access to Db2.
7. Restart the queue managers.
8. Recover the IBM MQ definitions from backup copies.
9. Reload any offloaded messages to the shared queues.

When the queue managers restart, they attempt to resynchronize local COPY objects with the Db2 GROUP objects. This might cause IBM MQ to attempt to do the following:

- Create COPY objects for old GROUP objects that existed at the point in time Db2 has recovered to.
- Delete COPY objects for GROUP objects that were created since the point in time Db2 has recovered to and so do not exist in the database.

The DELETE of COPY objects is attempted with the NOPURGE option, so it fails for queue managers that still have messages on these COPY queues.

Db2 and the coupling facility fail

If the coupling facility fails, the queue manager might fail, and Db2 will also fail if it is using this coupling facility.

Recover Db2 using Db2 recovery procedures. When Db2 has been restarted, you can restart the queue managers. The CF administration structure will also have failed, but this is rebuilt by restarting all the queue managers within the queue-sharing group.

If a single application structure within the coupling facility suffers a failure, the effect on the queue manager depends on the level of the queue manager and the CFLEVEL of the failed CF structure:

- If the CF application structure is CFLEVEL(3) or higher and RECOVER is set to YES, it will not be usable until you recover the CF structure by issuing an MQSC RECOVER CFSTRUCT command to the queue manager that will do the recovery. You can specify a single CF structure to be recovered, or you can recover several CF structures simultaneously. The queue manager performing the recovery locates the relevant backups on all the other queue managers' logs using the data in Db2 and the bootstrap data sets. The queue manager replays these backups in the correct time sequence across the queue sharing group, from just before the last backup through to the point of failure. If a recoverable application structure has failed, any further application activity is prevented until the structure has been recovered. If the administration structure has also failed, all the queue managers in the queue-sharing group must be started before the RECOVER CFSTRUCT command can be issued. All queue managers can continue working with local queues and queues in other CF structures during recovery of a failed CF structure.
- If the CF application structure is CFLEVEL(3) or higher and RECOVER is set to NO, the structure is automatically reallocated by the next **MQOPEN** request performed on a queue defined in the structure. All messages are lost, as the structure can only contain non-persistent messages.
- If the CF application structure has a CFLEVEL less than 3, the queue manager fails. On queue manager restart, peer recovery attempts to connect to the structure, detect that the structure has failed and allocate a new version of the structure. All messages on shared queues that were in CF structures affected by the coupling facility failure are lost.

Since IBM WebSphere MQ Version 7.1, queue managers in queue-sharing-groups have been able to tolerate loss of connectivity to coupling facility structures without failing. If the structure has experienced a connection failure, attempts are made to rebuild the structure in another coupling facility with better connectivity in order to regain access to shared queues as soon as possible.

Problems with long-running units of work

Use this topic to investigate, and resolve problems with long-running units of work.

This topic explains what to do if you encounter a long-running unit of work during restart. In this context, this means a unit of work that has been active for a long time (possibly days or even weeks) so that the origin RBA of the unit of work is outside the scope of the current active logs. This means that restart could take a long time, because all the log records relating to the unit of work have to be read, which might involve reading archive logs.

Old unit of work found during restart

Problem
: A unit of work with an origin RBA that predates the oldest active log has been detected during restart.

Symptoms
: IBM MQ issues the following message:

```
CSQR020I +CSQ1 OLD UOW FOUND
```

System action
: Information about the unit of work is displayed, and message CSQR021D is issued, requesting a response from the operator.

System programmer action
: None.

Operator action
: Decide whether to commit the unit of work or not. If you choose not to commit the unit of work, it is handled by normal restart recovery processing. Because the unit of work is old, this is likely to involve using the archive log, and so takes longer to complete.

IMS-related problems

Use this topic to investigate, and resolve problems with IMS and IBM MQ..

This topic includes plans for the following problems that you might encounter in the IMS environment:

- “IMS cannot connect to IBM MQ”
- “IMS application problem”
- “IMS is not operational” on page 1870

IMS cannot connect to IBM MQ

Problem
: The IMS adapter cannot connect to IBM MQ.

Symptoms

IMS remains operative. The IMS adapter issues these messages for control region connect:

- CSQQ001I
- CSQQ002E
- CSQQ003E
- CSQQ004E
- CSQQ005E
- CSQQ007E

For details, see the IBM MQ for z/OS messages, completion, and reason codes documentation.

If an IMS application program tries to access IBM MQ while the IMS adapter cannot connect, it can either receive a completion code and reason code, or terminate abnormally. This depends on the value of the REO option in the SSM member of IMS PROCLIB.

System action

All connection errors are also reported in the IMS message DFS3611.

System programmer action

None.

Operator action

Analyze and correct the problem, then restart the connection with the IMS command:

```
/START SUBSYS subsysname
```

IMS requests the adapter to resolve in-doubt units of recovery.

IMS application problem

Problem

An IMS application terminates abnormally.

Symptoms

The following message is sent to the user's terminal:

```
DFS555I  TRANSACTION tran-id ABEND abcode
MSG IN PROCESS: message data:
```

where *tran-id* represents any IMS transaction that is terminating abnormally and *abcode* is the abend code.

System action
IMS requests the adapter to resolve the unit of recovery. IMS remains connected to IBM MQ.

System programmer action
None.

Operator action
As indicated in message DFS554A on the IMS master terminal.

IMS is not operational

Problem
IMS is not operational.

Symptoms
More than one symptom is possible:

- IMS waits or loops

  IBM MQ cannot detect a wait or loop in IMS, so you must find the origin of the wait or loop. This can be IMS, IMS applications, or the IMS adapter.
- IMS terminates abnormally.
  - See the manuals *IMS/ESA Messages and Codes* and *IMS/ESA Failure Analysis Structure Tables* for more information.
  - If threads are connected to IBM MQ when IMS terminates, IBM MQ issues message CSQ3201E. This message indicates that IBM MQ end-of-task (EOT) routines have been run to clean up and disconnect any connected threads.

System action
IBM MQ detects the IMS error and:

- Backs out in-flight work.
- Saves in-doubt units of recovery to be resolved when IMS is reconnected.

System programmer action
None.

Operator action
Resolve and correct the problem that caused IMS to terminate abnormally, then carry out an emergency restart of IMS. The emergency restart:

- Backs out in-flight transactions that changed IMS resources.
- Remembers the transactions with access to IBM MQ that might be in doubt.

You might need to restart the connection to IBM MQ with the IMS command:

```
/START SUBSYS subsysname
```

During startup, IMS requests the adapter to resolve in-doubt units of recovery.

Hardware problems

Use this topic as a starting point to investigate hardware problems.

If a hardware error causes data to be unreadable, IBM MQ can still be recovered by using the *media recovery* technique:

1. To recover the data, you need a backup copy of the data. Use DFDSS or Access Method Services REPRO regularly to make a copy of your data.
2. Reinstate the most recent backup copy.
3. Restart the queue manager.

The more recent your backup copy, the more quickly your subsystem can be made available again.

When the queue manager restarts, it uses the archive logs to reinstate changes made since the backup copy was taken. You must keep sufficient archive logs to enable IBM MQ to reinstate the changes fully. Do not delete archive logs until there is a backup copy that includes all the changes in the log.

Index

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

Trademarks

Sending your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:
- Send an email to ibmkc@us.ibm.com
- Use the form on the web here: www.ibm.com/software/data/rcf/

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

Include the following information:
- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number
- The topic and page number related to your comment
- The text of your comment

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

Thank you for your participation.