IBM WebSphere MQ

Administering IBM WebSphere MQ

Version 7 Release 5

Note
Before using this information and the product it supports, read the information in "Notices" on page 1213.

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

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

Contents

Figures

Tables

Administering IBM WebSphere 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® WebSphere® MQ objects locally or remotely, see "Local and remote administration" on page 3.

There are a number of different methods that you can use to create and administer your queue managers and their related resources in IBM WebSphere 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 WebSphere MQ depending on your platform:

- "IBM WebSphere MQ control commands"
- "IBM WebSphere MQ Script (MQSC) commands"
- "Programmable Command Formats (PCFs)" on page 2

There are also the other following options for creating and managing IBM WebSphere MQ objects:

- "The IBM WebSphere MQ Explorer" on page 2
- "The Windows Default Configuration application" on page 3
- "The Microsoft Cluster Service (MSCS)" on page 3

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 WebSphere MQ Administration Interface (MQAI) on some platforms. For more information about automating administration tasks, see "Automating administration tasks" on page 4.

IBM WebSphere MQ control commands

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

IBM WebSphere 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. For the command reference for the control commands, see WebSphere MQ Control commands WebSphere MQ Control commands.

IBM WebSphere 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. 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 MQSC reference.

"Script (MQSC) Commands" on page 69 contains a description of each MQSC command and its syntax.

See "Performing local administration tasks using MQSC commands" on page 69 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 WebSphere 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 5.

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

The IBM WebSphere MQ Explorer

Using the IBM WebSphere 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 IBM WebSphere MQ Explorer by using the system menu, the `MQExplorer` executable file, or the **strmqcfg** command.

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

You can use IBM WebSphere 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.

See "Administration using the IBM WebSphere MQ Explorer" on page 52 for more information.

The Windows Default Configuration application

You can use the Windows Default Configuration program to create a *starter* (or default) set of IBM WebSphere 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 WebSphere MQ clusters. The distinction is:

IBM WebSphere 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 WebSphere MQ for Windows system to use MSCS.

Related concepts:

"Administering local WebSphere MQ objects" on page 65
This section tells you how to administer local WebSphere 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 WebSphere MQ objects.

"Administering remote WebSphere MQ objects" on page 102

Related information:

WebSphere MQ technical overview

Planning

Configuring

Transactional support scenarios

Considerations when contact is lost with the XA resource manager

Local and remote administration

You can administer WebSphere 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 WebSphere MQ, you can consider this as local administration because no channels are involved, that is, the communication is managed by the operating system.

WebSphere 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 WebSphere 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 WebSphere MQ Explorer.

"Administering remote WebSphere MQ objects" on page 102 describes the subject of remote administration in greater detail.

How to use WebSphere MQ control commands

This section describes how to use the IBM WebSphere 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 WebSphere MQ on UNIX, Linux and Windows systems. In addition, note the following environment-specific information:

WebSphere 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.

WebSphere MQ for UNIX and Linux systems
: All WebSphere 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 WebSphere MQ Explorer.

For more information, see The WebSphere 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 WebSphere MQ objects.

PCF commands

WebSphere 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 a WebSphere 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 WebSphere 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 WebSphere MQ for AIX®
- IBM WebSphere MQ for HP-UX
- IBM WebSphere MQ for Linux

- IBM WebSphere MQ for Solaris
- IBM WebSphere MQ for Windows
- 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.

WebSphere 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 WebSphere 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 WebSphere MQ remote administration.

This section includes:

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.

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 WebSphere MQ COPY, header, include and module files.

Message descriptor for a PCF command

The WebSphere 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 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 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 WebSphere MQ Windows Explorer is one such application (however, the Version 6.0 or later WebSphere 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.

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 WebSphere 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.

Windows UNIX Linux

WebSphere 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, WebSphere MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 13.

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

WebSphere 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, WebSphere MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 13.

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

WebSphere MQ Object authorities

Table 1. Windows, HP Integrity NonStop Server, UNIX and Linux systems - object authorities

Command	**WebSphere 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

Table 1. Windows, HP Integrity NonStop Server, UNIX and Linux systems - object authorities (continued)

Command	**WebSphere MQ object authority**	**Class authority (for object type)**
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
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

Table 1. Windows, HP Integrity NonStop Server, UNIX and Linux systems - object authorities (continued)

Command	WebSphere MQ object authority	Class authority (for object type)
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*
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.

WebSphere 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.

Using the MQAI to simplify the use of PCFs

The MQAI is an administration interface to WebSphere 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.

Use the MQAI in the following ways:

To simplify the use of PCF messages

The MQAI is an easy way to administer WebSphere MQ; you do not have to write your own PCF messages, avoiding 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 do this, you need several statements in your program for every structure, and memory space must be allocated. 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 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.

To handle error conditions more easily
It is difficult to get return codes back from PCF commands, but 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, which 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 WebSphere MQ Administration Interface (MQAI)."

Introduction to the WebSphere MQ Administration Interface (MQAI)

WebSphere MQ Administration Interface (MQAI) is a programming interface to WebSphere MQ. It performs administration tasks on a WebSphere 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 WebSphere 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 a WebSphere 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 42. For more information about PCFs, see "Introduction to Programmable Command Formats" on page 5

Use of the MQAI

You can use the MQAI to:

- Simplify the use of PCF messages. The MQAI is an easy way to administer WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ and to exchange data between applications, see "Using the MQAI to simplify the use of PCFs" on page 15.

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)"
2. Display events on the screen using a simple event monitor. "Displaying events using an event monitor (amqsaiem.c)" on page 22
3. Print a list of all local queues and their current depths. "Inquiring about queues and printing information (amqsailq.c)" on page 36
4. Print a list of all channels and their types. "Inquire channel objects (amqsaicl.c)" on page 30

Building your MQAI application

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

Hints and tips for configuring WebSphere 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 WebSphere MQ using the MQAI can be found in "Hints and tips for configuring WebSphere MQ" on page 41

WebSphere MQ Administration Interface (MQAI)

WebSphere MQ for Windows, AIX, Linux, HP-UX, and Solaris support the WebSphere MQ Administration Interface (MQAI). The MQAI is a programming interface to WebSphere 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 WebSphere 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          */
/*               WebSphere 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 WebSphere 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,                    /* WebSphere 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 WebSphere 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 needs 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 WebSphere 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 WebSphere 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 WebSphere 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,                  /* WebSphere 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 WebSphere MQ

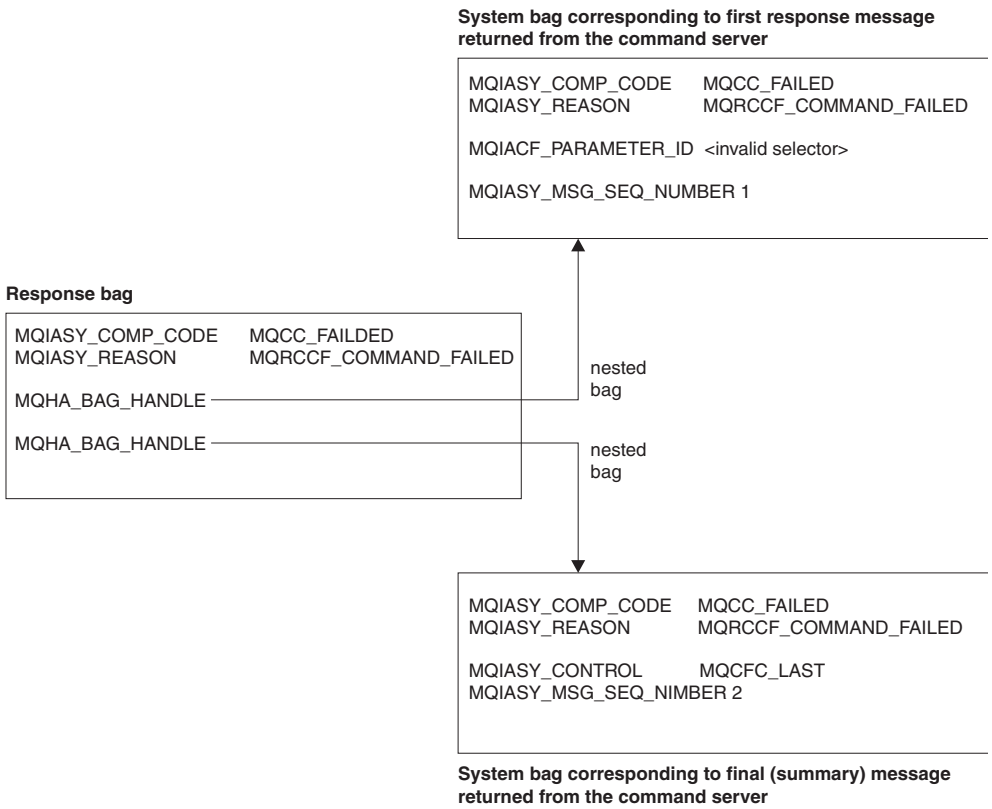
Programming hints and tips when using MQAI.

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 WebSphere MQ using the MQAI:

- Character strings in WebSphere MQ are blank padded to a fixed length. Using C, null-terminated strings can normally be supplied as input parameters to WebSphere 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 51

  The following diagram shows this scenario:



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 45

  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 the these concepts is shown in Figure 1.

The hierarchy has been explained in a previous paragraph.

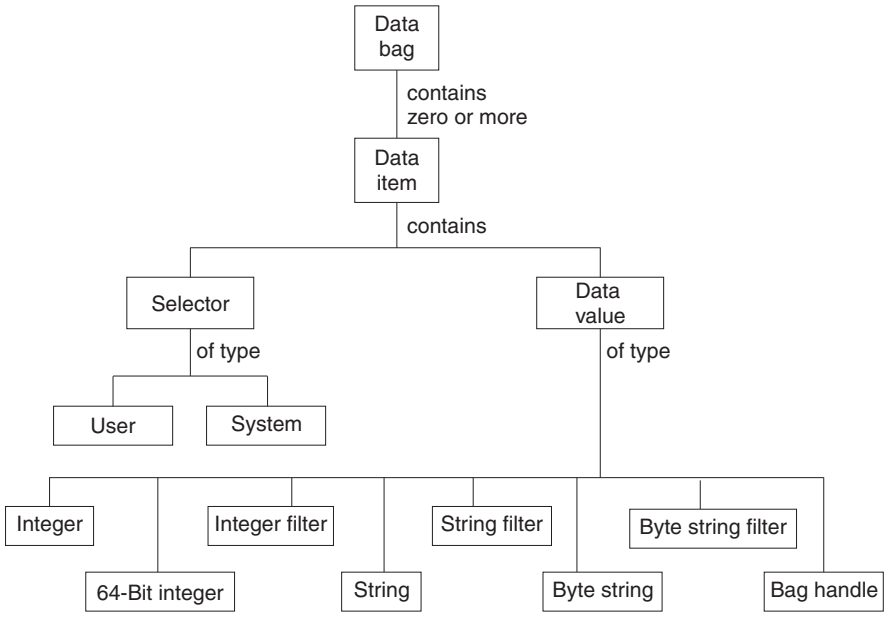


Figure 1. Hierarchy of MQAI concepts

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 WebSphere 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 44.

command bag
A bag also created for commands for administering WebSphere 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" on page 44.

group bag
A bag used to hold a set of grouped data items. Group bags cannot be used for administering WebSphere 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" on page 44.
- You can send data between applications using data bags "Putting and receiving data bags" on page 44.

- You can add data items to data bags “Adding data items to bags” on page 46.
- You can add an inquiry command within a data bag “Adding an inquiry command to a bag” on page 46.
- You can inquire within data bags “Inquiring within data bags” on page 47.
- You can count data items within a data bag “Counting data items” on page 50.
- You can change information within a data bag “Changing information within a bag” on page 48.
- You can clear a data bag “Clearing a bag using the mqClearBag call” on page 49.
- You can truncate a data bag “Truncating a bag using the mqTruncateBag call” on page 49.
- You can convert bags and buffers “Converting bags and buffers” on page 49.

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 45.
- 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 51. 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 8

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.”

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 50.
- “Deleting data items” on page 50.
- “Adding data items to bags” on page 46.
- “Filtering and querying data items” on page 47.

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 48), but the bag-creation options cannot be changed by the user. You cannot delete system items. (See “Deleting data items” on page 50.)

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 45.

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 45.

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 WebSphere 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 WebSphere 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 5. 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 17.

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).

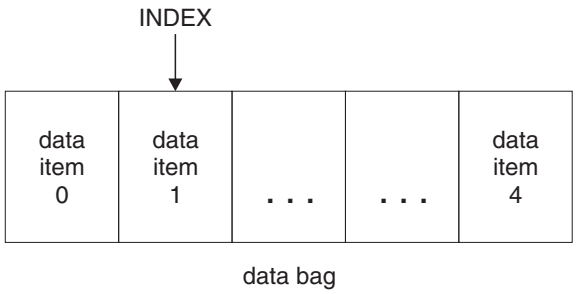


Figure 3. Modifying a single data item

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.

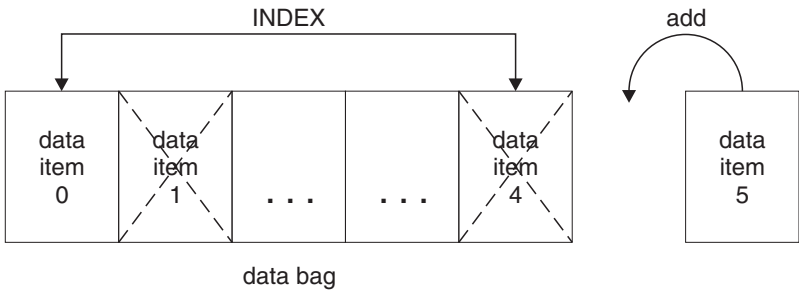


Figure 4. Modifying all data items

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.

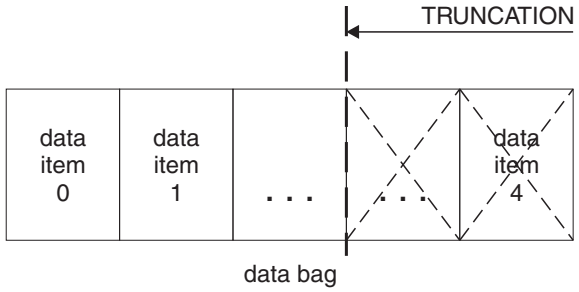


Figure 5. Truncating a bag

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.

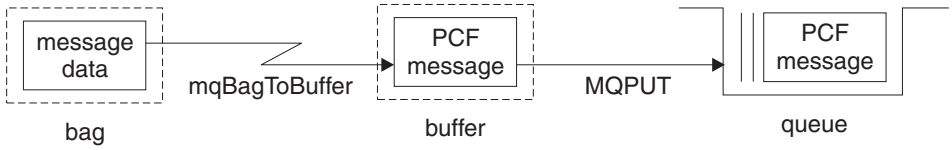


Figure 6. Converting bags to PCF messages

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 on page 50. For a full description of the mqBufferToBag call, see mqBufferToBag.

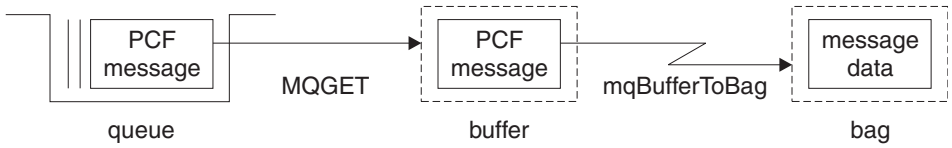


Figure 7. Converting PCF messages to bag form

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."
- 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 49.
- 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 49.

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.)

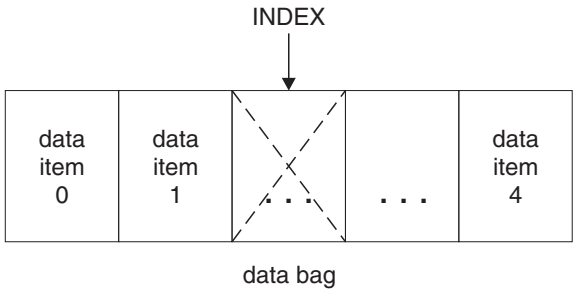


Figure 8. Deleting a single data item

or

2. All occurrences of the specified selector. (See Figure 9 on page 51.)

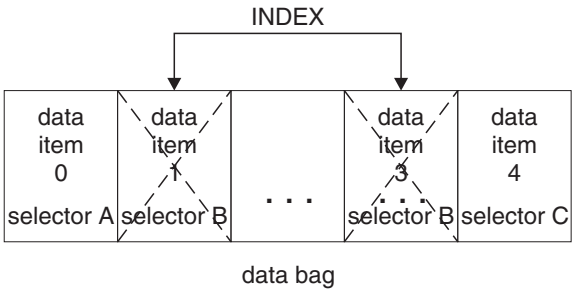


Figure 9. Deleting all data items

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 WebSphere 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 on page 52.

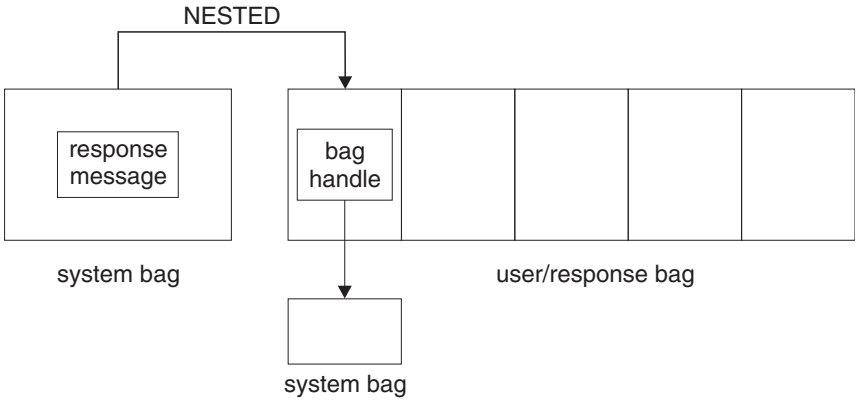


Figure 10. Nesting

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 IBM WebSphere MQ Explorer

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

IBM WebSphere MQ for Windows, and IBM WebSphere MQ for Linux (x86 and x86-64 platforms) provide an administration interface called the IBM WebSphere 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 IBM WebSphere MQ Explorer.

The IBM WebSphere MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows, or Linux (x86-64 platforms), by pointing the IBM WebSphere MQ

Explorer at the queue managers and clusters you are interested in. The platforms and levels of IBM WebSphere MQ that can be administered using the IBM WebSphere MQ Explorer are described in "Remote queue managers" on page 54.

To configure remote IBM WebSphere MQ queue managers so that IBM WebSphere MQ Explorer can administer them, see "Prerequisite software and definitions" on page 55.

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

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

On Linux, to start the IBM WebSphere 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 WebSphere MQ Explorer

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

With the WebSphere 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 WebSphere 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 WebSphere MQ objects, and WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Explorer can connect to all supported queue managers with the following exceptions:

- WebSphere MQ for z/OS queue managers earlier than Version 6.0.
- Currently supported MQSeries® V2 queue managers.

The WebSphere 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 WebSphere MQ Explorer on a WebSphere MQ V5.3 computer, you must install Fix Pack 9 (CSD9) or later on your WebSphere MQ for Windows V5.3 computer.

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

Deciding whether to use the WebSphere MQ Explorer

When deciding whether to use the WebSphere 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 WebSphere MQ Explorer, when you work with the objects using MQSC commands, you must enclose the object names in single quotation marks, or WebSphere MQ does not recognize them.

Large queue managers
The WebSphere 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 WebSphere MQ Explorer extracts the required information to present in a view.

Clusters
WebSphere MQ clusters can potentially contain hundreds or thousands of queue managers. The WebSphere 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 WebSphere MQ Explorer dramatically because the WebSphere MQ Explorer does not connect to the queue managers in the cluster until you select them.

Setting up the WebSphere MQ Explorer

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

Prerequisite software and definitions

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

The IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Explorer.

Authorization to use the WebSphere MQ Explorer:

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

To perform local administrative tasks using the WebSphere 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 WebSphere MQ Explorer, the user executing the WebSphere 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 WebSphere MQ for z/OS and perform remote administrative tasks using the WebSphere 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 WebSphere 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 WebSphere MQ objects, see Giving access to a WebSphere 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 WebSphere MQ Explorer is to display all WebSphere MQ objects. If there are any WebSphere 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 WebSphere MQ Explorer and each remote queue manager.

The IBM WebSphere 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 WebSphere MQ Explorer:

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

You can define a default security exit, which can be used for all new client connections from the WebSphere 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 WebSphere MQ Explorer. For more information, see the WebSphere MQ Help Center.

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

The IBM WebSphere 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 WebSphere MQ MQI clients.

When you have established the channel using a client channel definition table, you can use the IBM WebSphere 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 IBM WebSphere MQ Explorer."

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 IBM WebSphere 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\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 IBM WebSphere MQ Explorer

On the system hosting the IBM WebSphere 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 IBM WebSphere MQ Explorer uses Java keystore files (JKS) for SSL security, and so the keystore file being created for configuring SSL for the IBM WebSphere 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 IBM WebSphere MQ Explorer toolbar, click **Window -> Preferences**, then expand **WebSphere 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 IBM WebSphere MQ Explorer" on page 57, 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 57 on the system hosting the remote queue manager.
9. Click **Finish**. You can now access the remote queue manager from the IBM WebSphere MQ Explorer.

Connecting through another queue manager:

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

In this case, the WebSphere 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.

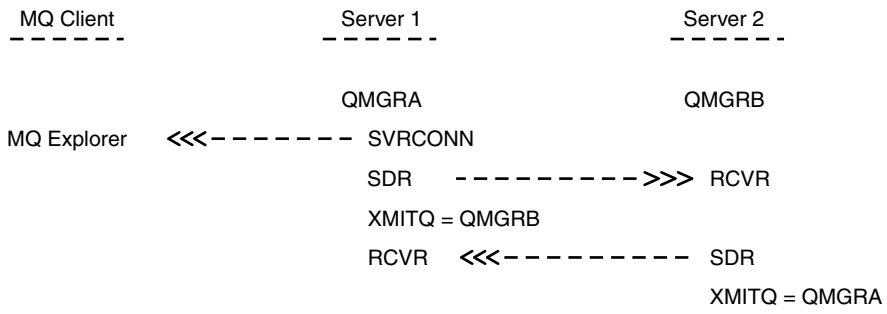
If the connection is then used to connect to the target queue manager via an intermediate queue manager, the userId is flowed in the *UserIdentifier* parameter of the message descriptor (MQMD) again. In order for the MCA listener on the target queue manager to accept this message, either the MCAUSER 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 WebSphere 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:

- WebSphere 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 WebSphere MQ Explorer through an intermediate queue manager (QMGRA)
- When connecting to QMGRB with WebSphere MQ Explorer, select QMGRA as the intermediate queue manager

In this situation, there is no direct connection to QMGRB from WebSphere 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 105.

Showing and hiding queue managers and clusters

The WebSphere 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 WebSphere 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 WebSphere MQ** tree node in the Navigation pane of the WebSphere 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 WebSphere MQ** tree node in the Navigation pane of the WebSphere 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 WebSphere MQ Explorer.

Cluster membership

WebSphere 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 WebSphere MQ Explorer is running, then you must maintain the WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Explorer connects to local queue managers that are members of a cluster directly, without using a client connection.

Data conversion

The WebSphere MQ Explorer works in CCSID 1208 (UTF-8). This enables the WebSphere 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 WebSphere 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 WebSphere MQ Explorer and a queue manager with a CCSID that the WebSphere MQ Explorer does not recognize.

Supported conversions are described in Code page conversion.

Security on Windows

The Prepare WebSphere 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 a IBM WebSphere MQ installation. One service is created for each installation. Each service is named `MQ_InstallationName`, and has a display name of `IBM WebSphere MQ(InstallationName)`. Before Version 7.1, with only one installation on a server the single, Windows service was named MQSeriesServices with the display name `IBM 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 a IBM WebSphere MQ Windows Service" on page 62

When you install IBM WebSphere MQ and run the Prepare IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ is running under might not have the authority it requires to query the group membership of other domain user accounts. The Prepare IBM WebSphere 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 WebSphere MQ is running under does not have the required authority, the Prepare IBM WebSphere 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 a IBM WebSphere MQ Windows Service" on page 62. Once the user has entered valid account details for the domain user account into the Prepare IBM WebSphere MQ Wizard, it configures a

IBM WebSphere 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, a IBM WebSphere MQ Windows service is launched and remains running for as long as the service is running. A IBM WebSphere MQ administrator who logs on to the server after the Windows service is launched can use the IBM WebSphere MQ Explorer to administer queue managers on the server. This connects the IBM WebSphere 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 WebSphere MQ administrator requires Access permission.

User rights required for a IBM WebSphere 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 a IBM WebSphere MQ installation runs.

Log on as batch job	Enables a IBM WebSphere MQ Windows service to run under this user account.
Log on as service	Enables users to set the IBM WebSphere MQ Windows service to log on using the configured account.
Shut down the system	Allows the IBM WebSphere 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 WebSphere MQ Service

You might need to change the user name associated with the IBM WebSphere 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 WebSphere MQ Wizard to enter the details of the new user account.

Changing the password of the IBM WebSphere MQ Windows service user account

About this task

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

Procedure

1. Identify the user the service is running under.
2. Stop the IBMIBM WebSphere 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 WebSphere 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 WebSphere MQ Windows service for an installation running under a domain user account: About this task

If the IBM WebSphere 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 IBMIBM WebSphere MQ service.

   The user account that IBM WebSphere 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 WebSphere MQ administration rights. By default it is added to the local **mqm** group on the server. If this membership is removed, the IBM WebSphere MQ Windows service does not work. For more information about user rights, see "User rights required for a IBM WebSphere MQ Windows Service" on page 62

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

Related concepts:

"Using Active directory (Windows only)" on page 61
In some network configurations, where user accounts are defined on domain controllers that are using Active Directory, the local user account IBM WebSphere MQ is running under might not have the authority it requires to query the group membership of other domain user accounts. The Prepare IBM WebSphere MQ Wizard identifies whether this is the case by carrying out tests and asking the user questions about the network configuration.

IBM WebSphere MQ coordinating with Db2 as the resource manager

If you start your queue managers from the IBM WebSphere MQ Explorer, or are using IBM WebSphere 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 theIBM WebSphere 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 WebSphere 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 WebSphere MQ Explorer

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

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

The WebSphere 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 WebSphere MQ Explorer, system administrators have the ability to customize the WebSphere MQ Explorer to improve the way they administer WebSphere MQ.

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

Using the WebSphere MQ Taskbar application (Windows only)

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

On Windows, the WebSphere 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; WebSphere 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 WebSphere MQ icon. From the menu you can perform the following actions:

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

The WebSphere MQ alert monitor application (Windows only)

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

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

From the WebSphere MQ alert monitor, you can:

- Access the WebSphere MQ Explorer directly
- View information relating to all outstanding alerts
- Shut down the WebSphere 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 WebSphere MQ objects

This section tells you how to administer local WebSphere 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 WebSphere MQ objects.

In addition to the approaches detailed in this section you can use the WebSphere MQ Explorer to administer local WebSphere MQ objects; see "Administration using the IBM WebSphere MQ Explorer" on page 52.

This section contains the following information:

- Application programs using the MQI
- "Performing local administration tasks using MQSC commands" on page 69
- "Working with queue managers" on page 77
- "Working with local queues" on page 79
- "Working with alias queues" on page 85
- "Working with model queues" on page 87
- "Working with services" on page 94
- "Managing objects for triggering" on page 100

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

1. Open the WebSphere 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 WebSphere 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 WebSphere MQ for Windows you can start a queue manager automatically when the system starts using the WebSphere MQ Explorer. For more information, see "Administration using the IBM WebSphere MQ Explorer" on page 52.

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

1. Open the WebSphere 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 67 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.

Stopping queue managers in WebSphere MQ for Windows

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

To stop a queue manager running under WebSphere 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)
AMQZDMAA	Deferred message processor
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
AMQZTRCN	Trace

3. Stop the WebSphere 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 WebSphere MQ for UNIX and Linux systems

How to end the processes and the WebSphere MQ service, to stop queue managers in WebSphere 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 WebSphere 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)
amqzdmaa	Deferred message processor
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, FFSTs 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.

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 WebSphere MQ for Windows or IBM WebSphere MQ for Linux (x86 and x86-64 platforms), you can also perform the operations described in this section using the IBM WebSphere MQ Explorer. See "Administration using the IBM WebSphere MQ Explorer" on page 52 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 information about administering 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."

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 73.)

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.

MQSC commands are summarized in the collection of topics in the MQSC reference section.

Script (MQSC) Commands

MQSC commands provide a uniform method of issuing human-readable commands on WebSphere MQ platforms. For information about *programmable command format* (PCF) commands, see "Introduction to Programmable Command Formats" on page 5.

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.
- 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).

- 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:
  - Generic values ending with an asterisk
  - A single asterisk (for example, TRACE(*))
  - 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.
- 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 WebSphere MQ, see “Performing local administration tasks using MQSC commands” on page 69.

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.

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

Related information:

Building command scripts

WebSphere 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 theMQSC reference for more information.

The **runmqsc** command invocation, in common with all WebSphere MQ control commands, is case sensitive in some WebSphere 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 WebSphere 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: WebSphere 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 the MQSC reference 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 71 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 74 shows an extract from the MQSC command file `myprog.in` and Figure 12 on page 75 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 74 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its attributes. The MQSC reference 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 WebSphere 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 75. 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: WebSphere 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: WebSphere 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 WebSphere MQ:

amqscos0.tst
Definitions of objects used by sample programs.

amqscic0.tst
Definitions of queues for CICS® transactions.

In WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

If you cannot get MQSC commands to run, use the following information 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 generated.

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: WebSphere 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: WebSphere 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 78

```
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(750)                           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 the MQSC reference 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 80.
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 82.

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 tell the queue manager about the dead-letter queue.

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)

WebSphere 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 Handling undelivered messages with the WebSphere MQ dead-letter queue handler.

Resolving problems with undelivered messages:

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 Handling undelivered messages with the WebSphere MQ dead-letter queue handler.

Displaying default object attributes

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

When you define a WebSphere 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 79, 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

WebSphere 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 WebSphere MQ is installed.

In WebSphere 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 WebSphere 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 84.

```
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

Enabling large queues

IBM WebSphere 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 WebSphere MQ website for platform-specific performance reports:

`http://www.ibm.com/software/integration/ts/mqseries/`

Working with alias queues

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

V 7.5.0.8

Attention: Distribution lists do not support the use of alias queues that point to topic objects. From Version 7.5.0, Fix Pack 8, if an alias queue points to a topic object in a distribution list, IBM WebSphere 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 79).
- A local definition of a remote queue (see "Creating a local definition of a remote queue" on page 110).
- 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 the MQSC reference for more information about this and other alias queue commands.

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 reference 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 89
Use the MQSC command **DISPLAY TOPIC** to display an administrative topic object.

"Changing administrative topic attributes" on page 89
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 90
You can copy a topic definition using the LIKE attribute on the **DEFINE** command.

"Deleting an administrative topic definition" on page 90
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 about topic trees, 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" on page 89.
- 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 89

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 about topic trees, 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 the MQSC reference 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 92
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 93
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 93
You can copy a subscription definition using the **LIKE** attribute on the **DEFINE** command.

"Deleting a subscription" on page 93
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 description 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" on page 92.
- If you already have a local subscription on the same queue manager with the name TEST, 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 93.
- 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: WebSphere 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: WebSphere 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: WebSphere 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 92.
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 82.

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 WebSphere 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:\Program Files\IBM\WebSphere 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 WebSphere 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       : WebSphere 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 WebSphere MQ is installed.

MQ_DATA_PATH
The location of the WebSphere MQ data directory:

- On UNIX and Linux systems, the WebSphere MQ data directory location is `/var/mqm/`
- On Windows systems, the location of the WebSphere MQ data directory is the data directory selected during the installation of WebSphere 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 WebSphere 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."

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 WebSphere 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 97 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 WebSphere 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

WebSphere 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 WebSphere 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.

Administering remote WebSphere MQ objects

This section tells you how to administer WebSphere 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

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 WebSphere 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 WebSphere 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 105 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 a WebSphere 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.

For more information about clusters, their attributes, and how to set them up, see Queue manager clusters.

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 6.

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 WebSphere 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 on page 105 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.

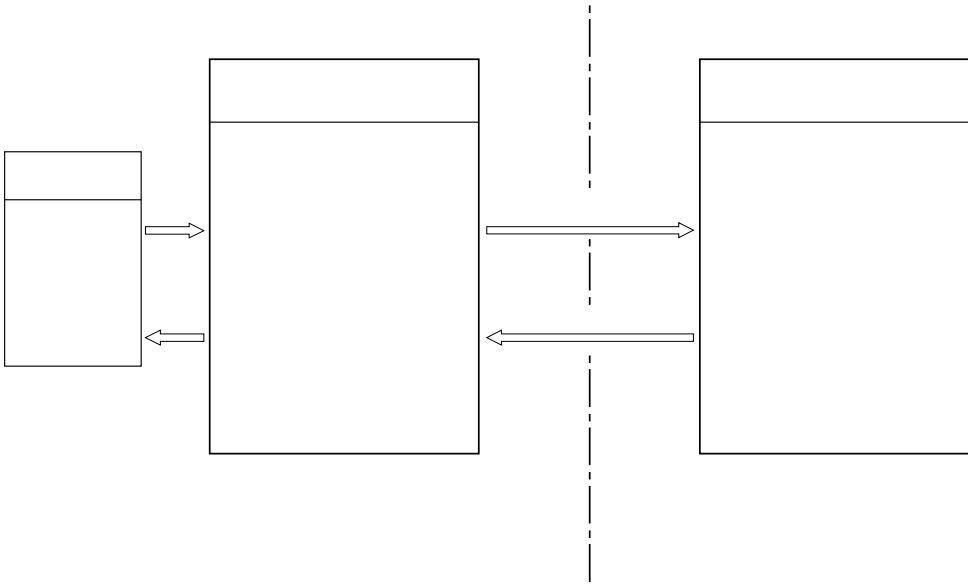


Figure 16. Remote administration using MQSC commands

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 on page 106 summarizes this configuration.

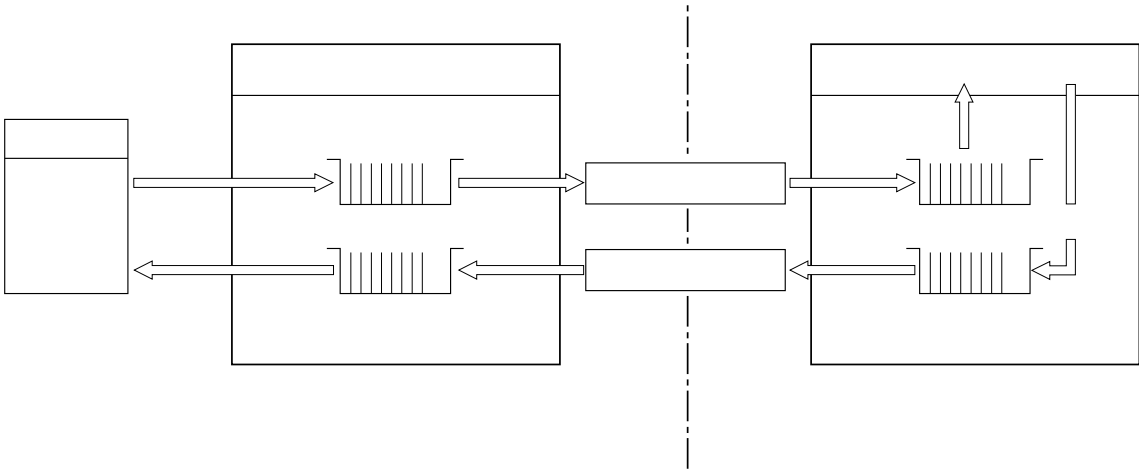


Figure 17. Setting up channels and queues for remote administration

See Connecting applications using 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 WebSphere 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 WebSphere 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 Auto-definition of cluster 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 WebSphere MQ for Windows or WebSphere MQ for Linux (x86 and x86-64 platforms) can perform the operations described in the following sections using the WebSphere MQ Explorer. For more information, see "Administration using the IBM WebSphere MQ Explorer" on page 52.

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 107.

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 76.

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 110.

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ for Windows, ccsid.tbl is located in directory `C:\Program Files\IBM\WebSphere MQ\conv\table` by default.

In WebSphere 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, WebSphere 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 WebSphere MQ object names (as defined in Naming IBM WebSphere 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 WebSphere MQ Telemetry

IBM WebSphere MQ Telemetry is administered using IBM WebSphere 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 WebSphere MQ. Configure the security of IBM WebSphere MQ Telemetry using JAAS, SSL and the IBM WebSphere MQ object authority manager.

Administering using IBM WebSphere MQ Explorer

Use the explorer to configure telemetry channels, control the telemetry service, and monitor the MQTT clients that are connected to IBM WebSphere MQ. Configure the security of IBM WebSphere MQ Telemetry using JAAS, SSL and the IBM WebSphere MQ object authority manager.

Administering using the command line

IBM WebSphere MQ Telemetry can be completely administered at the command line using the IBM WebSphere MQ MQSC commands.

The IBM WebSphere MQ Telemetry documentation also has sample scripts that demonstrate the basic usage of the MQ Telemetry Transport v3 Client application.

Read and understand the samples in MQ Telemetry Transport sample programs in the Developing applications for IBM WebSphere MQ Telemetry section before using them.

Related concepts:

"Configure distributed queuing to send messages to MQTT clients" on page 119
WebSphere 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`.

"MQTT client identification, authorization, and authentication" on page 121
To authorize an MQTT client to access WebSphere MQ objects, authorize the `ClientIdentifier`, or `Username` of the client, or authorize a common client identity. To permit a client to connect to WebSphere MQ, authenticate the `Username`, or use a client certificate. Configure JAAS to authenticate the `Username`, and configure SSL to authenticate a client certificate.

"Telemetry channel authentication using SSL" on page 128
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.

"Publication privacy on telemetry channels" on page 130
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.

"SSL configuration of MQTT clients and telemetry channels" on page 131
Configure SSL to authenticate the telemetry channel, the MQTT client, and encrypt the transfer of messages between clients and the telemetry channel.

"Telemetry channel JAAS configuration" on page 137
Configure JAAS to authenticate the `Username` sent by the client.

"WebSphere MQ Telemetry daemon for devices concepts" on page 139
The WebSphere MQ Telemetry daemon for devices is an advanced MQTT V3 client application. Use it to store and forward messages from other MQTT clients. It connects to WebSphere MQ like an MQTT client, but you can also connect other MQTT clients to it.

Related tasks:

"Configuring a queue manager for telemetry on Linux and AIX"
Follow these manual steps to configure a queue manager to run IBM WebSphere MQ Telemetry. You can run an automated procedure to set up a simpler configuration using the IBM WebSphere MQ Telemetry support for IBM WebSphere MQ Explorer.

"Configuring a queue manager for telemetry on Windows" on page 117
Follow these manual steps to configure a queue manager to run IBM WebSphere MQ Telemetry. You can run an automated procedure to set up a simpler configuration using the IBM WebSphere MQ Telemetry support for IBM WebSphere MQ Explorer.

Related information:

WebSphere 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 WebSphere MQ Telemetry. You can run an automated procedure to set up a simpler configuration using the IBM WebSphere MQ Telemetry support for IBM WebSphere MQ Explorer.

Before you begin

1. See Installing IBM WebSphere MQ Telemetry for information on how to install IBM WebSphere MQ, and the IBM WebSphere 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 WebSphere MQ Telemetry support for IBM WebSphere 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 WebSphere MQ Telemetry by using IBM WebSphere MQ Explorer and MQ Telemetry Transport sample programs.

Follow the steps in this task to configure IBM WebSphere 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 IBM WebSphere MQ Explorer or with the command in Figure 19 on page 118.

   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 WebSphere MQ messages; see "Sending a message to a client directly" on page 120.
4. Follow a procedure in "Authorizing MQTT clients to access WebSphere MQ objects" on page 123 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 IBM WebSphere 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 WebSphere MQ Telemetry. You can run an automated procedure to set up a simpler configuration using the IBM WebSphere MQ Telemetry support for IBM WebSphere MQ Explorer.

Before you begin

1. See Installing IBM WebSphere MQ Telemetry for information on how to install IBM WebSphere MQ, and the IBM WebSphere 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 WebSphere MQ Telemetry support for IBM WebSphere 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 WebSphere MQ Telemetry by using IBM WebSphere MQ Explorer and MQ Telemetry Transport sample programs.

Follow the steps in this task to configure IBM WebSphere 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 IBM WebSphere 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 WebSphere MQ messages; see "Sending a message to a client directly" on page 120.

4. Follow a procedure in "Authorizing MQTT clients to access WebSphere MQ objects" on page 123 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 IBM WebSphere 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 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

WebSphere 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 WebSphere 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 WebSphere MQ object name resolution, *ClientIdentifier* is resolved as follows; see Table 2.

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

Table 2. Name resolution of an MQTT queue manager alias (continued)

	Input		Output		
ClientIdentifier	**Queue manager name**	**Queue name**	**Queue manager name**	**Queue name**	**Transmission 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 WebSphere 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 WebSphere 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 WebSphere MQ applications, can.

The WebSphere 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, WebSphere MQ topics on behalf of MQTT clients, using MQTT channels. The WebSphere MQ administrator configures the MQTT channel identity that is used for WebSphere 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 WebSphere MQ objects, authorize the `ClientIdentifier`, or `Username` of the client, or authorize a common client identity. To permit a client to connect to WebSphere 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 `ClientIdentifier`, `Username`, or a common client identity for authorization to access WebSphere MQ objects.

The WebSphere 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 WebSphere MQ topics. The choices are:

1. The client identifier.
2. An identity the administrator provides for the channel.
3. 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.

Use the WebSphere 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, to authorize 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 WebSphere MQ objects:

Follow these steps to authorize MQTT clients to publish and subscribe to WebSphere MQ Objects. The steps follow four alternative access control patterns.

Before you begin

MQTT clients are authorized to access objects in WebSphere MQ by being assigned an identity when they connect to a telemetry channel. The WebSphere MQ Administrator configures the telemetry channel using WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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" on page 124
- "Coarse-grained access control" on page 124
- "Medium-grained access control" on page 124
- "Fine-grained access control" on page 124

The result of the models is to assign *`mqttUsers`* sets of permissions to publish and subscribe to WebSphere MQ, and receive publications from WebSphere MQ.

No access control:

MQTT clients are given WebSphere 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 WebSphere 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 PublishX -all +pub
setmqaut -m qMgr -t topic -n topic1 -p SubscribeX -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 WebSphere 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 WebSphere MQ administrator sets the MQTT channel identity to any name, such as MQTTClientUser, using WebSphere MQ Explorer.
2. The WebSphere 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 WebSphere 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.

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 WebSphere 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 WebSphere 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 WebSphere MQ administrator.

You can also use SSL cipher suites for client authentication. Here is an alphabetic list of the SSL cipher suites that are currently supported:

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA`
- `SSL_KRB5_EXPORT_WITH_RC4_40_MD5`
- `SSL_KRB5_EXPORT_WITH_RC4_40_SHA`
- `SSL_KRB5_WITH_3DES_EDE_CBC_MD5`
- `SSL_KRB5_WITH_3DES_EDE_CBC_SHA`
- `SSL_KRB5_WITH_DES_CBC_MD5`
- `SSL_KRB5_WITH_DES_CBC_SHA`
- `SSL_KRB5_WITH_RC4_128_MD5`
- `SSL_KRB5_WITH_RC4_128_SHA`

- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 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 129
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
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

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 WebSphere 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. Here is an alphabetic list of the SSL cipher suites that are currently supported:

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_AES_128_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_RC4_128_SHA`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_AES_128_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5`
- `SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA`
- `SSL_KRB5_EXPORT_WITH_RC4_40_MD5`

- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V 7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V 7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V 7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V 7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V 7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V 7.5.0.2 If you plan to use SHA-2 cipher suites, see "System requirements for using SHA-2 cipher suites with MQTT channels" on page 812.

Related concepts:

"Telemetry channel configuration for MQTT client authentication using SSL" on page 126
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.

"CipherSpecs and CipherSuites" on page 175
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

Related reference:

../com.ibm.mq.ref.adm.doc/q085610_.dita
Syntax diagram for a telemetry channel when using the **DEFINE CHANNEL** command.

../com.ibm.mq.ref.adm.doc/q085260_.dita
Syntax diagram for a telemetry channel when using the ALTER CHANNEL command. This is separate from the regular ALTER CHANNEL syntax diagram and parameter descriptions.

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 128.

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 clients and telemetry channels

MQTT clients and the WebSphere MQ Telemetry (MQXR) service use Java Secure Socket Extension (JSSE) to connect telemetry channels using SSL. The WebSphere MQ Telemetry daemon for devices does not support SSL.

Configure SSL to authenticate the telemetry channel, the MQTT client, and encrypt the transfer of messages between clients and the telemetry channel.

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 125

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 128.
3. Encrypt messages between the client and the queue manager; see “Publication privacy on telemetry channels” on page 130.

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. WebSphere MQ Telemetry channel
2. MQTT Java client
3. JRE

Configure the telemetry channel parameters using WebSphere 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.

WebSphere MQ Telemetry channel

Set all the telemetry channel SSL parameters using WebSphere 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 WebSphere 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 WebSphere 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 on page 136 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 WebSphere 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 138.

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 WebSphere 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 WebSphere 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 138 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

WebSphere MQ Telemetry daemon for devices concepts

The WebSphere MQ Telemetry daemon for devices is an advanced MQTT V3 client application. Use it to store and forward messages from other MQTT clients. It connects to WebSphere MQ like an MQTT client, but you can also connect other MQTT clients to it.

The daemon is a publish/subscribe broker. MQTT V3 clients connect to it to publish and subscribe to topics, using topic strings to publish, and topic filters to subscribe. The topic string is hierarchical, with topic levels divided by /. Topic filters are topic strings that can include single level + wildcards and a multilevel # wildcard as the last part of the topic string.

Note: Wildcards in the daemon follow the more restrictive rules of WebSphere Message Broker, v6. WebSphere MQ is different. It supports mulitple multilevel wildcards; wildcards can stand in for any number of levels of the hierarchy, anywhere in the topic string.

Multiple MQTT v3 clients connect to the daemon using a listener port. The default listener port is modifiable. You can define multiple listener ports and allocate different namespaces to them, see "WebSphere MQ Telemetry daemon for devices listener ports" on page 146. The daemon is itself an MQTT v3 client. Configure a daemon bridge connection to connect the daemon to the listener port of another daemon, or to a WebSphere MQ Telemetry (MQXR) service.

You can configure multiple bridges for the WebSphere MQ Telemetry daemon for devices. Use the bridges to connect together a network of daemons that can exchange publications.

Each bridge can publish and subscribe to topics at its local daemon. It can also publish and subscribe to topics at another daemon, a WebSphere MQ publish/subscribe broker, or any other MQTT v3 broker it is connected to. Using a topic filter, you can select the publications to propagate from one broker to another. You can propagate publications in either direction. You can propagate publicaitons from the local daemon to each of its attached remote brokers, or from any of the attached brokers to the local daemon; see "WebSphere MQ Telemetry daemon for devices bridges."

WebSphere MQ Telemetry daemon for devices bridges

A WebSphere MQ Telemetry daemon for devices bridge connects two publish/subscribe brokers using the MQTT v3 protocol. The bridge propagates publications from one broker to the other, in either direction. At one end is a WebSphere MQ Telemetry daemon for devices bridge connection, and at the other might be a queue manager, or another daemon. A queue manager is connected to the bridge connection using a telemetry channel. A daemon is connected to the bridge connection using a daemon listener.

WebSphere MQ Telemetry daemon for devices supports one or more simultaneous connections to other brokers. The connections from the daemon are called bridges and are defined by connection entries in the daemon configuration file. The connections to WebSphere MQ are made using WebSphere MQ telemetry channels, as shown in the following figure:

Figure 30. Connecting IBM WebSphere MQ Telemetry daemon for devices to IBM WebSphere MQ

A bridge connects the daemon to another broker as an MQTT v3 client. The bridge parameters mirror the attributes of an MQTT v3 client.

A bridge is more than a connection. It acts as a publish and subscribe agent situated between two publish/subscribe brokers. The local broker is the WebSphere MQ Telemetry daemon for devices, and the remote broker is any publish/subscribe broker that supports the MQTT v3 protocol. Typically the remote broker is another daemon or WebSphere MQ.

The job of the bridge is to propagate publications between the two brokers. The bridge is bidirectional. It propagates publications in either direction. Figure 30 on page 139 illustrates the way the bridge connects WebSphere MQ Telemetry daemon for devices to WebSphere MQ. "Example topic settings for the bridge" uses examples to illustrate how to use the topic parameter to configure the bridge.

The In and Out arrows in Figure 30 on page 139 indicate the bidirectionality of the bridge. At one end of the arrow, a subscription is created. The publications that match the subscription are published to the broker at the opposite end of the arrow. The arrow is labeled according to the flow of publications. Publications flow In to the daemon and Out from the daemon. The importance of the labels is they are used in the command syntax. Remember that In and Out refer to where the publications flow, and not to where the subscription is sent.

Other clients, applications, or brokers might be connected either to WebSphere MQ or to WebSphere MQ Telemetry daemon for devices. They publish and subscribe to topics at the broker they are connected to. If the broker is WebSphere MQ, the topics might be clustered or distributed, and not explicitly defined at the local queue manager.

Uses of bridges

Connect daemons together using bridge connections and listeners. Connect daemons and queue managers together using bridge connections and telemetry channels. When you connect multiple brokers together it is possible to create loops. Be careful: Publications might circulate endlessly around a loop of brokers, undetected.

Some of the reasons for using daemons bridged to WebSphere MQ are as follows:

Reduce the number of MQTT client connections to WebSphere MQ
Using a hierarchy of daemons you can connect many clients to WebSphere MQ; more clients than the number a single queue manager can connect at one time.

Store and forward messages between MQTT clients and WebSphere MQ
You might use store and forward to avoid maintaining continuous connections between clients and WebSphere MQ, if the clients do not have their own storage. You might use multiple types of connections between the MQTT client and WebSphere MQ; see Telemetry concepts and scenarios for monitoring and control.

Filter the publications exchanged between MQTT clients and WebSphere MQ
Commonly, publications divide into messages that are processed locally and messages that involve other applications. Local publications might include control flows between sensors and actuators, and remote publications include requests for readings, status, and configuration commands.

Change the topic spaces of publications
Avoid topics strings from clients attached to different listener ports from colliding with one another. The example uses the daemon to label meter readings coming from different buildings; see Separating the topic spaces of different groups of clients.

Example topic settings for the bridge

Publish everything to the remote broker - using defaults

The default direction is called out, and the bridge publishes topics to the remote broker. The topic parameter controls what topics are propagated using topic filters.

The bridge uses the topic parameter in Figure 31 on page 141 to subscribe to everything published to the local daemon by MQTT clients, or by other brokers. The bridge publishes the topics to the remote broker connected by the bridge.

```
connection Daemon1
topic #
```

Figure 31. Publish everything to the remote broker

Publish everything to the remote broker - explicit

The topic setting in the following code fragment gives the same result as using the defaults. The only difference is that the **direction** parameter is explicit. Use the `out` direction to subscribe to the local broker, the daemon, and publish to the remote broker. Publications created on the local daemon that the bridge has subscribed to, are published at the remote broker.

```
connection Daemon1
topic # out
```

Figure 32. Publish everything to the remote broker - explicit

Publish everything to the local broker

Instead of using the direction `out`, you can set the opposite direction, `in`. The following code fragment configures the bridge to subscribe to everything published at the remote broker connected by the bridge. The bridge publishes the topics to the local broker, the daemon.

```
connection Daemon1
topic # in
```

Figure 33. Publish everything to the local broker

Publish everything from the export topic at the local broker to the import topic at the remote broker

Use two additional topic parameters, **local_prefix** and **remote_prefix**, to modify the topic filter, # in the previous examples. One parameter is used to modify the topic filter used in the subscription, and the other parameter is used to modify the topic the publication is published to. The effect is to replace the beginning of the topic string used in one broker with another topic string on the other broker.

Depending on the direction of the topic command the meaning of **local_prefix** and **remote_prefix** reverses. If the direction is `out`, the default, **local_prefix** is used as part of the topic subscription, and **remote_prefix** replaces the **local_prefix** part of the topic string in the remote publication. If the direction is `in`, **remote_prefix** becomes part of the remote subscription, and **local_prefix** replaces the **remote_prefix** part of the topic string.

The first part of a topic string is often thought of as defining a topic space. Use the additional parameters to change the topic space a topic is published to. You might do this to avoid the topic being propagated colliding with another the topic on the target broker, or to remove a mount point topic string.

As an example, in the following code fragment, all the publications to the topic string `export/#` at the daemon are republished to `import/#` at the remote broker.

```
topic # out export/ import/
```

Figure 34. Publish everything from the export topic at the local broker to the import topic at the remote broker

Publish everything to the import topic at the local broker from the export topic at the remote broker

The following code fragment shows the configuration reversed; the bridge subscribes to everything published with the `export/#` topic string at the remote broker and publishes it to `import/#` at the local broker.

```
connection Daemon1
topic # in import/ export/
```

Figure 35. Publish everything to the import topic at the local broker from the export topic at the remote broker

Publish everything from the 1884/ mount point to the remote broker with the original topic strings

In the following code fragment, the bridge subscribes to everything published by clients connected to the mount point `1884/` at the local daemon. The bridge publishes everything published to the mount point to the remote broker. The mount point string `1884/` is removed from the topics published to the remote broker. The *local_prefix* is the same as the mount point string `1884/`, and the *remote_prefix* is a blank string.

```
listener 1884
mount_point 1884/
connection Daemon1
topic # out 1884/ ""
```

Figure 36. Publish everything from the 1884/ mount point to the remote broker with the original topic strings.

Separating the topic spaces of different clients connected to different daemons

An application is written for electrical power meters to publish meter readings for a building. The readings are published using MQTT clients to a daemon hosted in the same building. The topic selected for the publications is `power`. The same application is deployed to a number of buildings in a complex. For site monitoring and data storage, readings from all buildings are aggregated using bridge connections. The connections link the building daemons to WebSphere MQ at a central location.

The client applications in each building are identical, but the data must be differentiated by building. Each reading has a `power` topic and must be prefixed with the building number to distinguish it. The bridge from the first building in the complex uses the prefix `meters/building01/`, from building two the prefix is `meters/building02/`. The readings from the other buildings follow the same pattern. WebSphere MQ receives the readings with topics like `meters/building01/power`.

The example is contrived; in practice the topic space the application publishes to is likely to be configurable.

The configuration file for each daemon has a topic statement that follows the pattern in the following code fragment:

```
connection Daemon1
topic power out "" meters/building01/
```

Figure 37. Separate the topic spaces of clients connected to different daemons

Specify an empty string as a placeholder for the unused local_prefix parameter.

Separate the topic spaces of clients connected to the same daemon

Suppose that a single daemon is used to connect all the power meters. Assuming that in the application can be configured to connect to different ports, you might distinguish the buildings by attaching the meters from different buildings to different listener ports, as in the following code fragment. Again, the example is contrived; it illustrates how mount points might be used.

```
listener 1884
mount_point meters/building01/
listener 1885
mount_point meters/building02/
connection Daemon1
topic meters/+/power out
```

Figure 38. Separate the topic spaces of clients connected to the same daemon

Remap different topics for publications flowing in both directions

In the configuration in the following code fragment, the bridge subscribes to the single topic `b` at the remote broker and forwards publications about `b` to the local daemon, changing the topic to `a`. The bridge also subscribes to the single topic `x` at the local broker and forwards publications about `x` to the remote broker, changing the topic to `y`.

```
connection Daemon1
topic "" in a b
topic "" out x y
```

Figure 39. Remap different topics for publications flowing in both directions

An important point about this example is that different topics are subscribed to and published to at both brokers. The topics spaces at both brokers are disjoint.

Remap the same topics for publications flowing in both directions (looping)

Unlike the previous example, the configuration in Figure 40 on page 144, in general, results in a loop. In the topic statement `topic "" in a b`, the bridge subscribes to `b` remotely, and publishes to `a` locally. In the other topic statement, the bridge subscribes to `a` locally, and publishes to `b` remotely. The same configuration can be written as shown in Figure 41 on page 144.

The general result is that if a client publishes to `b` remotely, the publication is transferred to the local daemon as a publication on topic `a`. However, on being published by the bridge to the local daemon on the topic `a`, the publication matches the subscription made by the bridge to local topic `a`. The subscription is `topic "" out a b`. As a result, the publication is transferred back to the remote broker as a publication on topic `b`. The bridge is now subscribed to the remote topic `b`, and the cycle begins again.

Some brokers implement loop detection to prevent the loop happening. But the loop detection mechanism must work when different types of brokers are bridged together. Loop detection does not work if WebSphere MQ is bridged to the WebSphere MQ Telemetry daemon for devices. It

does work if two WebSphere MQ Telemetry daemon for devices are bridged together. By default loop detection is turned on; see try_private.

```
connection Daemon1
topic "" in a b
topic "" out a b
```

Figure 40. !Remap the same topics for publications flowing in both directions

```
connection Daemon1
topic "" both a b
```

Figure 41. !Remap the same topics for publications flowing in both directions, using `both`.

The configuration in Figure 39 on page 143 is the same as Figure 40.

Availability of IBM WebSphere MQ Telemetry daemon for devices bridge connections

Configure multiple IBM WebSphere MQ Telemetry daemon for devices bridge connection addresses to connect to the first available remote broker. If the broker is a multi-instance queue manager, provide both of its TCP/IP addresses. Configure a primary connection to connect, or reconnect, to the primary server, when it is available.

The connection bridge parameter, addresses, is a list of TCP/IP socket addresses. The bridge attempts to connect to each address in turn, until it makes a successful connection. The round_robin and start_type connection parameters control how the addresses are used once a successful connection has been made.

If start_type is auto, manual, or lazy, then if the connection fails, the bridge attempts to reconnect. It uses each address in turn, with about a 20 second delay between each connection attempt. If `start_type` is once, then if the connection fails, the bridge does not attempt to reconnect automatically.

If round_robin is true, the bridge connection attempts start at the first address in the list and tries each address in the list in turn. It starts at the first address again, when the list is exhausted. If there is only one address in the list, it tries it again every 20 seconds.

If round_robin is false, the first address in the list, which is called the primary server, is given preference. If the first attempt to connect to the primary server fails, the bridge continues to try to reconnect to the primary server in the background. At the same time, the bridge tries to connect using the other addresses in the list. When the background attempts to connect to the primary server succeed, the bridge disconnects from the current connection, and switches to the primary server connection.

If a connection is disconnected voluntarily, for example by issuing a **`connection_stop`** command, then if the connection is restarted, it tries to use the same address again. If the connection is disconnected due to a failure to connect, or to the remote broker dropping the connection, the bridge waits 20 seconds. It then tries to connect to the next address in the list, or the same address, if there is only one address in the list.

Connecting to a multi-instance queue manager

In a multi-instance queue manager configuration, the queue manager runs on two different servers with different IP addresses. Typically telemetry channels are configured without a specific IP address. They are configured only with a port number. When the telemetry channel is started, by default it selects the first available network address on the local server.

Configure the addresses parameter of the bridge connection with the two IP addresses used by the queue manager. Set round_robin to true.

If the active queue manager instance fails, the queue manager switches over to the standby instance. The daemon detects that the connection to the active instance has broken and tries to reconnect to the standby instance. It uses the other IP address in the list of addresses configured for the bridge connection.

The queue manager to which the bridge connects is still the same queue manager. The queue manager recovers its own state. If cleansession is set to false, the bridge connection session is restored to the same state as before the failover. The connection resumes after a delay. Messages with "at least once" or "at most once" quality of service are not lost, and subscriptions continue to work.

The reconnection time depends on the number of channels and clients that restart when the standby instance starts, and how many messages were inflight. The bridge connection might try to reconnect to both IP addresses a number of times before the connection is reestablished.

Do not configure a multi-instance queue manager telemetry channel with a specific IP address. The IP address is only valid on one server.

If you are using an alternative high-availability solution, that manages the IP address, then it might be correct to configure a telemetry channel with a specific IP address.

cleansession

A bridge connection is an MQTT v3 client session. You can control whether a connection starts a new session, or whether it restores an existing session. If it restores an existing session, the bridge connection preserves the subscriptions and retained publications from the previous session.

Do not set cleansession to false if addresses lists multiple IP addresses, and the IP addresses connect to telemetry channels hosted by different queue managers, or to different telemetry daemons. Session state is not transferred between queue managers or daemons. Trying to restart an existing session on a different queue manager or daemon results in a new session being started. In-doubt messages are lost, and subscriptions might not behave as expected.

notifications

An application can keep track of whether the bridge connection is running by using notifications. A notification is a publication that has the value `1`, connected, or `0`, disconnected. It is published to *topicString* defined by the `notification_topic` parameter. The default value of *topicString* is **$SYS/broker/connection/***clientIdentifier***/state**. The default *topicString* contains the prefix `$SYS`. Subscribe to topics beginning with `$SYS` by defining a topic filter beginning with `$SYS`. The topic filter `#`, subscribe to everything, does not subscribe to topics beginning with `$SYS` on the daemon. Think of `$SYS` as defining a special system topic space distinct from the application topic space.

Notifications enable IBM WebSphere MQ Telemetry daemon for devices to notify MQTT clients when a bridge is connected or disconnected.

keepalive_interval

The keepalive_interval bridge connection parameter sets the interval between the bridge sending a TCP/IP ping to the remote server. The default interval is 60 seconds. The ping prevents the TCP/IP session being closed by the remote server, or by a firewall, that detects a period of inactivity on the connection.

clientid

A bridge connection is an MQTT v3 client session and has a clientIdentifier that is set by the bridge connection parameter clientid. If you intend reconnections to resume a previous session by setting the cleansession parameter to false, the clientIdentifier used in each session must be the same. The default value of clientid is *hostname.connectionName*, which remains the same.

Installation, verification, configuration, and control of the WebSphere MQ Telemetry daemon for devices

Installation, configuration, and control of the daemon is file-based.

Install the daemon by copying the Software Development Kit to the device where you are going to run the daemon .

As an example, run the MQTT client utility and connect to the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker; see Publish a message to a specific MQTT v3 client.

Configure the daemon by creating a configuration file; see WebSphere MQ Telemetry daemon for devices configuration file.

Control a running daemon by creating commands in the file, `amqtdd.upd`. Every 5 seconds the daemon reads the file, runs the commands, and deletes the file; see WebSphere MQ Telemetry daemon for devices command file.

WebSphere MQ Telemetry daemon for devices listener ports

Connect MQTT V3 clients to the WebSphere MQ Telemetry daemon for devices using listener ports. You can qualify a listener port with a mount point and a maximum number of connections.

A listener port must correspond to the port number specified on the MQTT client connect(serverURI) method of a client connecting to this port. It defaults on both the client and the daemon to `1883`.

You can change the default port for the daemon by setting the global definition `port` in the daemon configuration file. You can set specific ports by adding a `listener` definition to the daemon configuration file.

For each listener port, other than the default port, you can specify a mount point to isolate clients. Clients connected to a port with a mount point are isolated from other clients; see "WebSphere MQ Telemetry daemon for devices mount points" on page 147.

You can limit the number of clients that can connect to any port. Set the global definition `max_connections` to limit connections to the default port, or qualify each listener port with `max_connections`.

Example

An example of a configuration file that changes the default port from `1883` to `1880`, and limits connections to port `1880` to `10000`. Connections to port `1884` are limited to `1000`. Clients attached to port `1884` are isolated from clients attached to other ports.

```
port 1880
max_connections 10000
listener 1884
mount_point 1884/
max_connections 1000
```

WebSphere MQ Telemetry daemon for devices mount points

You can associate a mount point with a listener port used by MQTT clients to connect to a WebSphere MQ Telemetry daemon for devices. A mount point isolates the publications and subscriptions exchanged by MQTT clients using one listener port from MQTT clients connected to a different listener port.

Clients attached to a listener port with a mount point can never directly exchange topics with clients attached to any other listener ports. Clients attached to a listener port without a mount point can publish or subscribe to topics of any client. Clients are not aware of whether they are attached through a mount point or not; it makes no difference to the topics strings created by clients.

A mount point is a string of text that is prefixed to the topic string of publications and subscriptions. It is prefixed to all the topic strings created by clients attached to listener port with a mount point. The string of text is removed from all topic strings sent to clients attached to the listener port.

If a listener port has no mount point, the topic strings of publications and subscriptions created and received by clients attached to the port are not altered.

Create mount point strings with a trailing `/`. That way the mount point is the parent topic of the topic tree for the mount point.

Example

A configuration file contains the following listener ports:

```
listener 1883
mount_point 1883/
listener 1884 127.0.0.1
mount_point 1884/
listener 1885
```

A client, attached to port `1883`, creates a subscription to `MyTopic`. The daemon registers the subscription as `1883/MyTopic`. Another client attached to port `1883` publishes a message on the topic, `MyTopic`. The daemon changes the topic string to `1883/MyTopic` and searches for matching subscriptions. The subscriber on port `1883` receives the publication with the original topic string `MyTopic`. The daemon has removed the mount point prefix from the topic string.

Another client, attached to port `1884`, also publishes on the topic `MyTopic`. This time the daemon registers the topic as `1884/MyTopic`. The subscriber on port `1883` does not receive the publication, because the different mount point results in a subscription with a different topic string.

A client, attached to port `1885`, publishes on the topic, `1883/MyTopic`. The daemon does not change the topic string. The subscriber on port `1883` receives the publication to `MyTopic`.

WebSphere MQ Telemetry daemon for devices quality of service, durable subscriptions and retained publications

Quality of service settings apply only to a running daemon. If a daemon stops, whether in a controlled manner, or because of a failure, the state of inflight messages is lost. The delivery of a message at least once, or at most once, cannot be guaranteed if the daemon stops. WebSphere MQ Telemetry daemon for devices supports limited persistence. Set the **`retained_persistence`** configuration parameter to save retained publications and subscriptions when the daemon is shut down.

Unlike WebSphere MQ, the WebSphere MQ Telemetry daemon for devices does not journal persistent data. Session state, message state, and retained publications are not saved transactionally. By default, the daemon discards all data when it stops. You can set an option to periodically checkpoint subscriptions and retained publications. Message status is always lost when the daemon stops. All non-retained publications are lost.

Set the daemon configuration option, `Retained_persistence` to `true`, to save retained publications periodically to a file. When the daemon restarts, the retained publications that were last autosaved are reinstated. By default, retained messages created by clients are not reinstated when the daemon restarts.

Set the daemon configuration option, `Retained_persistence` to `true`, to save subscriptions created in a persistent session periodically to a file. If `Retained_persistence` is set to `true`, subscriptions that clients create in a session with `CleanSession` set to `false`, a "persistent session", are restored. The daemon restores the subscriptions when it restarts, which start receiving publications. The client receives the publications when it restarts with `CleanSession` to `false`. By default, client session state is not saved when a daemon stops, and so subscriptions are not restored, even if the client sets `CleanSession` to `false`.

`Retained_persistence` is an autosave mechanism. It might not save the most recent retained publications or subscriptions. You can change how often retained publications and subscriptions are saved. Set the interval between saves, or the number of changes between saves, using the configuration options `autosave_on_changes` and `autosave_interval`.

Example configuration for setting persistence

```
# Sample configuration
# Daemon listens on port 1882 with persistence in /tmp
# Autosave every minute
port 1882
persistence_location /tmp/
retained_persistence true
autosave_on_changes false
autosave_interval 60
```

WebSphere MQ Telemetry daemon for devices security

The WebSphere MQ Telemetry daemon for devices can authenticate clients that connect to it, use credentials to connect to other brokers, and control access to topics. The security the daemon provides is limited by being built using the WebSphere MQ Telemetry C client, which does not provide SSL support. Consequently, connections to and from the daemon are not encrypted, and cannot be authenticated using certificates.

By default, no security is switched on.

Authentication of clients

MQTT clients can set a username and password using the methods MqttConnectOptions.setUserName and MqttConnectOptions.setPassword.

Authenticate a client that connects to the daemon by checking the username and password provided by a client against entries in the password file. To enable authentication, create a password file and set the `password_file` parameter in the daemon configuration file; see password_file.

Set the `allow_anonymous` parameter in the daemon configuration file to allow clients connecting without usernames or passwords to connect to a daemon that is checking authentication; see allow_anonymous. If a client does provide a username or password it is always checked against the password file, if the `password_file` parameter is set.

Set the `clientid_prefixes` parameter in the daemon configuration file to limit connections to specific clients. The clients must have `clientIdentifiers` that start with one of the prefixes listed in the `clientid_prefixes` parameter; see clientid_prefixes.

Bridge connection security

Each WebSphere MQ Telemetry daemon for devices bridge connection is an MQTT V3 client. You can set the username and password for each bridge connection as a bridge connection parameter in the daemon

configuration file; see username and password. A bridge can then authenticate itself to a broker.

Access control of topics

If clients are being authenticated, the daemon can also provide control access to topics for each user. The daemon grants access control based on matching the topic to which a client is either publishing or subscribing with an access topic string in the access control file; see acl_file.

The access control list has two parts. The first part controls access for all clients, including anonymous clients. The second part has a section for any user in the password file. It lists specific access control for each user.

Example

The security parameters are shown in the following example.

```
acl_file c:\WMQTDaemon\config\acl.txt
password_file c:\WMQTDaemon\config\passwords.txt
allow_anonymous true
connection Daemon1
username daemon1
password deamonpassword
```

Figure 42. Daemon configuration file

```
Fred:Fredpassword
Barney:Barneypassword
```

Figure 43. Password file, passwords.txt

```
topic home/public/#
topic read meters/#
user Fred
topic write meters/fred
topic home/fred/#
user Barney
topic write meters/barney
topic home/barney/#
```

Figure 44. Access control file, acl.txt

Administering multicast

Use this information to learn about the WebSphere 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 WebSphere MQ Multicast topics and communication information objects.

About this task

WebSphere 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.

WebSphere MQ Multicast topic topology

Use this example to understand the WebSphere MQ Multicast topic topology.

WebSphere 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`.

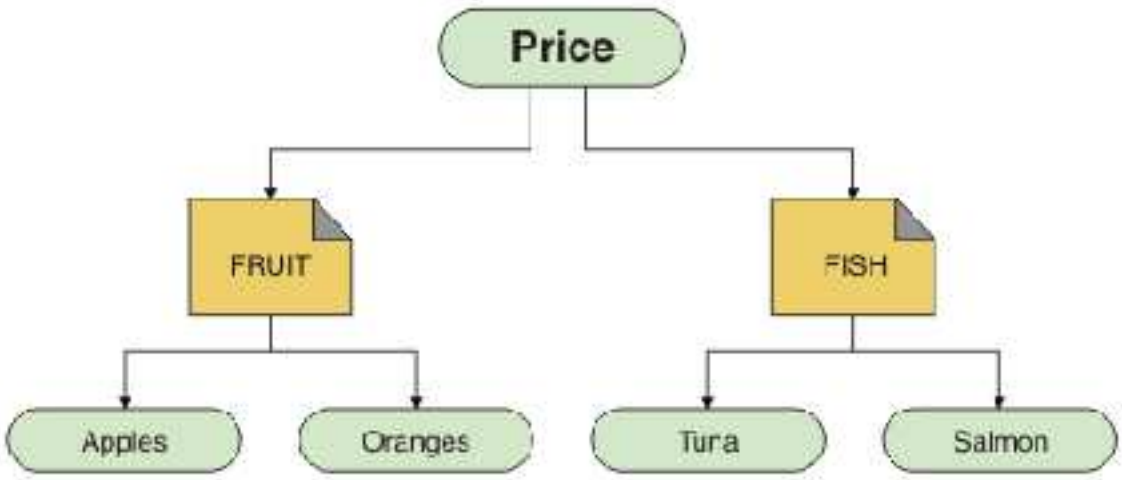
In the following diagram, there are two possible multicast data streams:

```
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.

WebSphere 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 WebSphere MQ message format, and reduce the size of WebSphere MQ messages.

WebSphere 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. WebSphere 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 153 .

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 WebSphere 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 WebSphere Message Broker RMM applications.

Multicast message attributes

Use this reference information to understand WebSphere 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
PutAppIType	Included if not default	MQAT_JAVA
PutAppIName	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 WebSphere MQ Multicast messaging.

WebSphere 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 WebSphere 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.

WebSphere 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 WebSphere 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 WebSphere MQ Multicast publishers and subscribers. The reliability indicator (the **MSGREL** field that is returned on the Publish or Subscribe status commands) is a WebSphere 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 WebSphere MQ Multicast subscription and message history.

A key element of overcoming transmission failure with multicast is WebSphere MQ's buffering of transmitted data (a history of messages to be kept at the transmitting end of the link). This process means that no buffering of messages is required in the putting application process because WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Multicast administration tasks such as configuring .ini files and interoperability with WebSphere 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 a WebSphere 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 WebSphere MQ Multicast fields in the .ini files.

Additional WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Low Latency Messaging

Use this information to understand the interoperability between WebSphere MQ Multicast and WebSphere 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 WebSphere 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 WebSphere 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. WebSphere MQ message properties to WebSphere MQ LLM property mappings

WebSphere MQ property	WebSphere 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: WebSphere MQ Low Latency Messaging.

Administering HP Integrity NonStop Server

Use this information to learn about administration tasks for the IBM WebSphere 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.

Security

Security is an important consideration for both developers of WebSphere MQ applications, and for system administrators configuring WebSphere MQ authorities.

Security overview

This collection of topics introduces the WebSphere 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 WebSphere MQ.

Security concepts and mechanisms

This collection of topics describes aspects of security to consider in your IBM WebSphere MQ installation.

The commonly accepted aspects of security are as follows:

- "Identification and authentication"
- "Authorization" on page 162
- "Auditing" on page 162
- "Confidentiality" on page 163
- "Data integrity" on page 163

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 163
- "Message digests and digital signatures" on page 165
- "Digital certificates" on page 166
- "Public Key Infrastructure (PKI)" on page 171

When you are planning a WebSphere 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 204.

Related concepts:

"Connecting two queue managers using SSL or TLS" on page 365
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.

"Working with SSL or TLS" on page 272
These topics give instructions for performing single tasks related to using SSL or TLS with IBM WebSphere MQ.

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 a IBM WebSphere MQ environment because IBM WebSphere 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 WebSphere MQ with IBM WebSphere 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 a WebSphere MQ environment by writing your own exit programs.

Related concepts:

"Identification and authentication in IBM WebSphere MQ" on page 178
In IBM WebSphere 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 WebSphere MQ" on page 178
You can use authorization to limit what particular individuals or applications can do in your IBM WebSphere 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 207 and the associated sub-topics.

Related concepts:

"Auditing in IBM WebSphere MQ" on page 178
IBM WebSphere 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 WebSphere MQ.

The term *entity* is used to refer to a queue manager, a WebSphere MQ MQI client, an individual user, or any other system capable of exchanging messages.

Related concepts:

"Cryptography in IBM WebSphere MQ" on page 179
IBM WebSphere 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*).

See the Glossary for a definition of cryptography.

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 176 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 45 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 46 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.

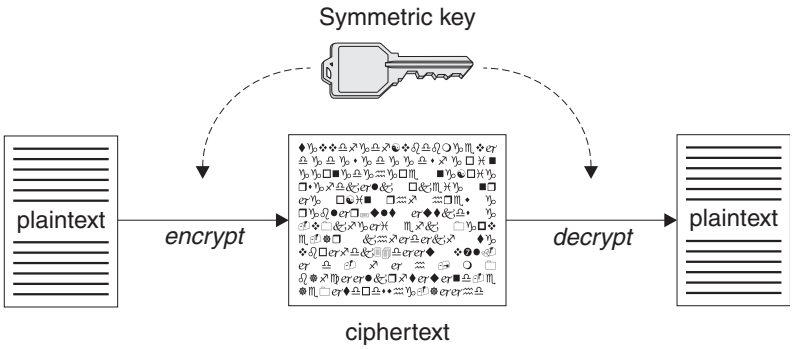


Figure 45. Symmetric key cryptography

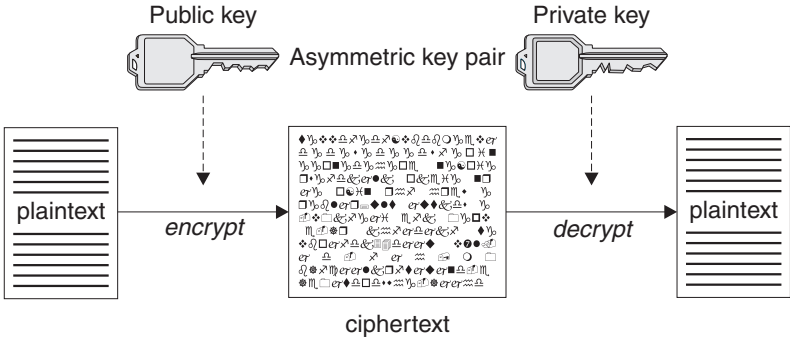


Figure 46. Asymmetric key cryptography

Figure 46 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, computed by a hash function. A message digest can be encrypted, forming a digital signature.

Messages are inherently variable in size. A message digest is a fixed size numeric representation of the contents of a message. A message digest is computed by a hash function, which is a transformation that meets two criteria:

- The hash function 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 176
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 167.

What is in a digital certificate:

Digital certificates contain specific pieces of information, as determined by the X.509 standard.

Digital certificates used by WebSphere 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 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:

WebSphere MQ supports digital certificates that comply with the X.509 standard. It requires the client authentication option.

Because IBM WebSphere 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 WebSphere MQ, see “Certificate validation policies in WebSphere MQ” on page 191.

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 309
- 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.

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 WebSphere MQ tolerates certain malformed DNs. For more information, see WebSphere MQ rules for SSLPEER values.

Related concepts:

"What is in a digital certificate" on page 166
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 the iKeyman tool on UNIX, Linux, and Windows systems and 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 47 on page 169 illustrates the process of obtaining a digital certificate from a CA.

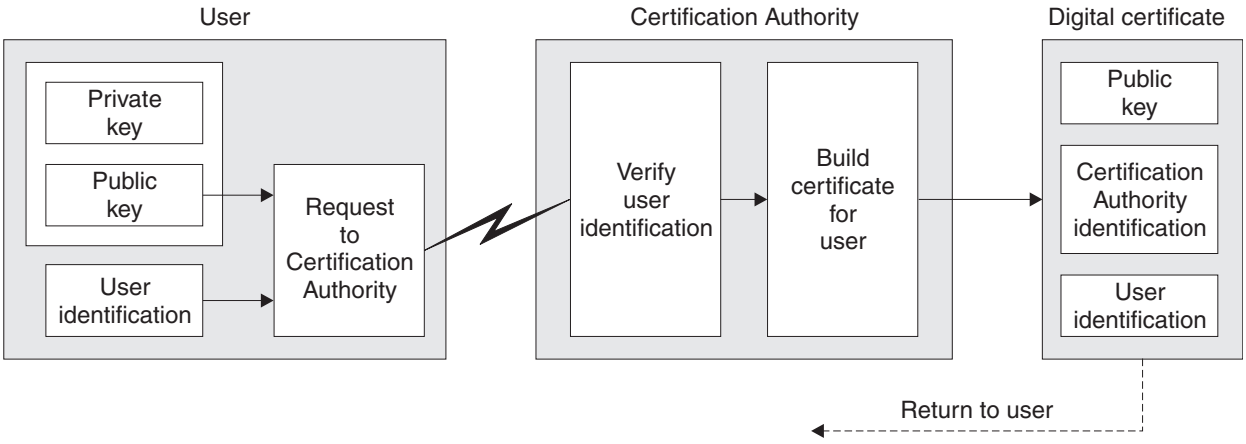


Figure 47. Obtaining a digital certificate

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 166.

Related information:

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 48 on page 170 illustrates a certification path from the certificate owner to the root CA, where the chain of trust begins.

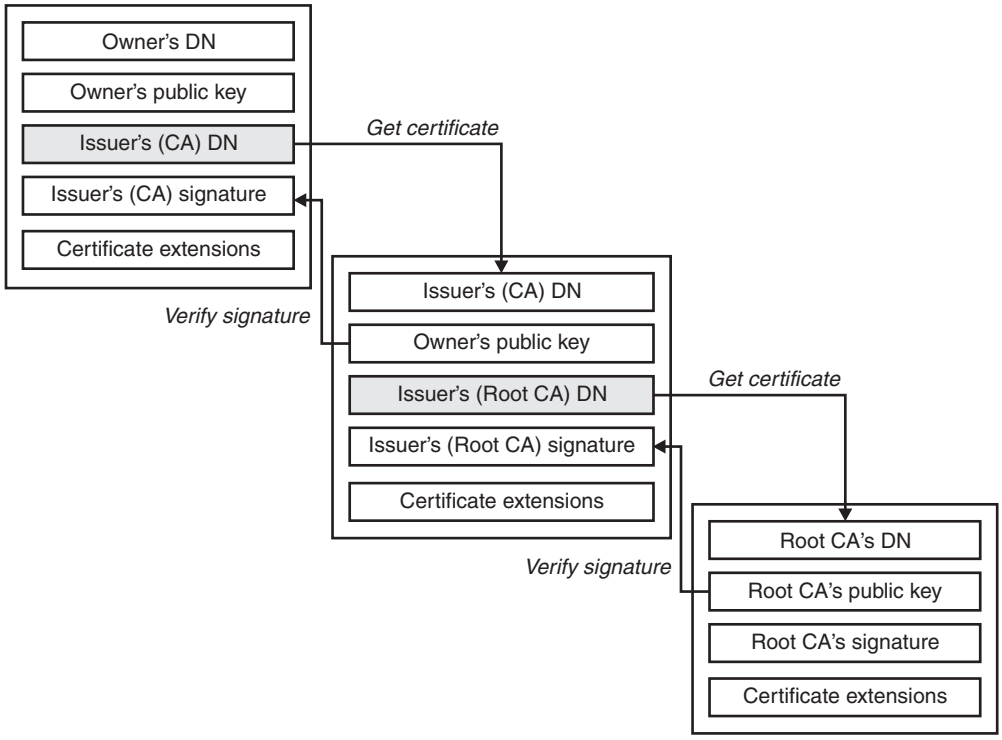


Figure 48. Chain of trust

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.

See the Glossary for a definition of certificate expiration.

Certificates can be revoked for various reasons, including:

- The owner has moved to a different organization.
- The private key is no longer secret.

WebSphere 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 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 309.

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 166 for more information about digital certificates and certificate authorities (CAs). RAs verify that 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 WebSphere 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:

"Security protocols in WebSphere MQ" on page 180
WebSphere 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 WebSphere MQ is provided at “Specifying CipherSpecs” on page 376.

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 173.
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 173.
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 49 illustrates the SSL or TLS handshake.

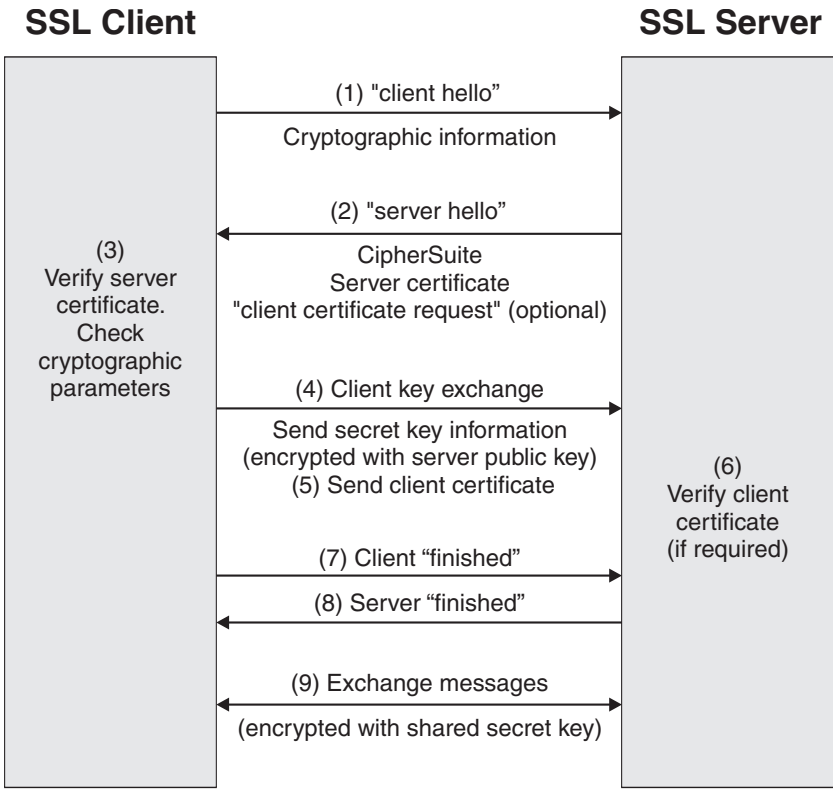


Figure 49. Overview of the SSL or TLS handshake

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 172 of the handshake. The exchange of finished messages that are encrypted with the secret key (steps 7 on page 172 and 8 on page 172 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 172 and 6 on page 172 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 176).
2. The certificate chain is checked; you should have intermediate CA certificates (see "How certificate chains work" on page 169).
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 309).

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 165 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 163.

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 385.

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 “Specifying CipherSpecs” on page 376.

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.

Important: When dealing with IBM WebSphere MQ channels, you use a CipherSpec. When dealing with Javachannels, JMS channels, or MQTT channels you specify a CipherSuite.

For more information about CipherSpecs, see “Specifying CipherSpecs” on page 376.

A CipherSuite is a suite of cryptographic algorithms used by an SSL or TLS 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 50 illustrates this process.

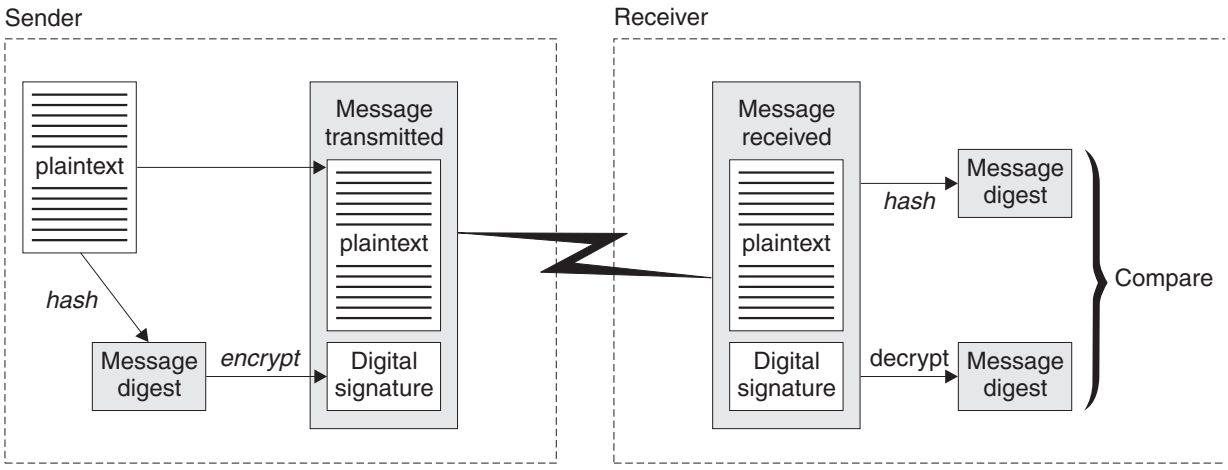


Figure 50. The digital signature process

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 WebSphere 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.

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 above will be 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, WebSphere 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 WebSphere MQ” on page 188.

Related information:

“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).

IBM WebSphere MQ security mechanisms

This collection of topics explains how you can implement the various security concepts in IBM WebSphere MQ.

IBM WebSphere MQ provides mechanisms to implement all the security concepts introduced in "Security concepts and mechanisms" on page 161. These are discussed in more detail in the following sections.

Identification and authentication in IBM WebSphere MQ

In IBM WebSphere MQ, you can implement identification and authentication using message context information and mutual authentication.

Here are some examples of the identification and authentication in a IBM WebSphere 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 161
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 WebSphere MQ

You can use authorization to limit what particular individuals or applications can do in your IBM WebSphere MQ environment.

Here are some examples of authorization in a IBM WebSphere MQ environment:

- Allowing only an authorized administrator to issue commands to manage IBM WebSphere 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 207 and the associated sub-topics.

Related concepts:

"Authorization" on page 162
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 WebSphere MQ

IBM WebSphere MQ can issue event messages to record that unusual activity has taken place.

Here are some examples of auditing in a IBM WebSphere 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 162
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 WebSphere MQ

You can implement confidentiality in IBM WebSphere MQ by encrypting messages.

Here are some examples of how confidentiality can be ensured in a IBM WebSphere MQ environment:

- After a sending MCA gets a message from a transmission queue, IBM WebSphere 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 WebSphere MQ might be considered sufficient to protect their contents against unauthorized disclosure. However, for a greater level of security, you can use IBM WebSphere MQ Advanced Message Security to encrypt the messages stored in the queues.

Related concepts:

"Confidentiality" on page 163
The *confidentiality* service protects sensitive information from unauthorized disclosure.

Data integrity in IBM WebSphere MQ

You can use a data integrity service to detect whether a message has been modified.

Here are some examples of how data integrity can be ensured in a IBM WebSphere MQ environment:

- 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 WebSphere MQ CipherSpecs provide a message digest algorithm, except for TLS_RSA_WITH_NULL_NULL which does not provide message data integrity.
- While messages are stored on a local queue, the access control mechanisms provided by IBM WebSphere 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 WebSphere 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.

Related concepts:

"Data integrity" on page 163
The *data integrity* service detects whether there has been unauthorized modification of data.

Cryptography in IBM WebSphere MQ

IBM WebSphere MQ provides cryptography by using the Secure sockets Layer (SSL) and Transport Security Layer (TLS) protocols.

For more information see "Security protocols in WebSphere MQ" on page 180.

Related concepts:

"Cryptographic concepts" on page 163
This collection of topics describes the concepts of cryptography applicable to WebSphere MQ.

Security protocols in WebSphere MQ

WebSphere 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. WebSphere MQ supports Version 3.0 of the SSL protocol and Version 1.0 and Version 1.2 of the Transport Layer Security (TLS) protocol. You specify the cryptographic algorithms that are used by the SSL or protocol by supplying a CipherSpec as part of the channel definition.

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 WebSphere MQ code at the client end of an MQI channel acts on behalf of the user of the WebSphere MQ client application. During the SSL or TLS handshake, the WebSphere MQ code sends the user's digital certificate to the MCA at the server end of the MQI channel.

Queue managers and WebSphere 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 a WebSphere MQ client system, the MQSSLKEYR environment variable specifies the location of the key repository that holds the user's digital certificate. Alternatively, a WebSphere 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.

Related concepts:

"Cryptographic security protocols: SSL and TLS" on page 171
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 WebSphere MQ supports both SSL and TLS.

WebSphere MQ support for SSL and TLS:

WebSphere MQ supports both the Secure Sockets Layer (SSL) protocol and the Transport Layer Security (TLS) protocol.

For more information about the SSL and TLS protocols, refer to the related information.

WebSphere MQ provides the following support for SSL Version 3.0 and TLS 1.0 and TLS 1.2:

Java and JMS clients
These clients use the JVM to provide SSL and TLS support.

Windows UNIX Linux **Windows, UNIX and Linux, and HP Integrity NonStop Server systems**
For UNIX, Linux, HP Integrity NonStop Server, and Windows systems, the SSL and TLS support is installed with WebSphere MQ.

For information about any prerequisites for WebSphere MQ SSL and TLS support, see IBM WebSphere MQ requirements.

The SSL or TLS key repository:

A mutually authenticated SSL or TLS connection requires a key repository (which can be known by different names on different platforms) 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 specific store names used on the platforms and environments that support SSL and TLS are:

Java and JMS	keystore and trust store
Windows UNIX Linux Windows , UNIX and Linux systems	key database file

For more information, refer to "Digital certificates" on page 166 and "Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts" on page 171.

A mutually authenticated SSL or TLS connection requires a key repository at each end of the connection. The key repository may contain:

- 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 WebSphere 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 WebSphere MQ key repositories" on page 182.

The location of the key repository depends on the platform you are using:

Windows UNIX Linux **Windows, UNIX and Linux systems**

On Windows, UNIX and Linux 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 WebSphere MQ is installed in the default location, the equivalent path on Windows is `C:\Program Files\IBM\WebSphere MQ\Qmgrs\QM1\ssl\key.kdb`.

On Windows , UNIX and Linux systems, 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: On Windows, UNIX and Linux systems, 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, WebSphere MQ still requires access to both a key database file and a password stash file.

On Windows and UNIX systems, the key database also contains the private key for the personal certificate associated with the queue manager or WebSphere MQ MQI client.

Protecting WebSphere MQ key repositories:

The key repository for WebSphere 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, WebSphere 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.

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 WebSphere MQ Explorer. For more information, see the MQCMD_REFRESH_SECURITY command and the topic *Refreshing SSL or TLS Security* in the WebSphere 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 a WebSphere 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 182
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.

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, for Windows, UNIX and Linux, and z/OS systems.

The FIPS 140-2 compliance of a WebSphere MQ SSL or TLS connection on UNIX, Linux, and Windows systems is found here "Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows."

If cryptographic hardware is present, the cryptographic modules used by WebSphere 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, WebSphere MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance. The WebSphere MQ 7.1 readme lists the version of FIPS enforced by each product maintenance level. If you configure WebSphere MQ to enforce FIPS compliance, always consult the readme when planning to apply maintenance. The readme is located at WebSphere MQ product READMEs

Related concepts:

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 269
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

"Using iKeyman, iKeycmd, runmqakm, and runmqckm" on page 275
On UNIX, Linux and Windows systems, manage keys and digital certificates with the iKeyman GUI or from the command line using iKeycmd or runmqakm.

Related information:

Enabling SSL in WebSphere MQ classes for Java

SSL properties of JMS objects

Using Secure Sockets Layer (SSL) with WebSphere MQ classes for JMS

"Federal Information Processing Standards" on page 177
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, WebSphere 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 a WebSphere MQ SSL or TLS connection on Windows, UNIX and Linux systems is as follows:

- For all WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 AIX, Linux, HP-UX, Solaris, Windows, and z/OS 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 WebSphere 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 WebSphere 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, WebSphere 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 382.

WebSphere 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 269
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

"Using iKeyman, iKeycmd, runmqakm, and runmqckm" on page 275
On UNIX, Linux and Windows systems, manage keys and digital certificates with the iKeyman GUI or from the command line using iKeycmd or runmqakm.

Related information:

Enabling SSL in WebSphere MQ classes for Java

SSL properties of JMS objects

Using Secure Sockets Layer (SSL) with WebSphere MQ classes for JMS

"Federal Information Processing Standards" on page 177
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 WebSphere MQ MQI client:

WebSphere MQ supports SSL and TLS on clients. You can tailor the use of SSL or TLS in various ways.

WebSphere MQ provides SSL and TLS support for WebSphere MQ MQI clients on Windows, UNIX and Linux systems. If you are using WebSphere MQ classes for Java, see Using WebSphere MQ classes for Java and if you are using WebSphere MQ classes for JMS, see Using WebSphere 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 a WebSphere MQ MQI client either with the MQSSLKEYR value in your WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ MQI clients. For more information, see "Resetting SSL and TLS secret keys" on page 382 and "Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows" on page 183.

See "Setting up WebSphere MQ MQI client security" on page 268 for more information about the SSL support for WebSphere 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 WebSphere 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 a WebSphere 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 WebSphere 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 a WebSphere MQ MQI client" on page 381
You have three options for specifying a CipherSpec for a WebSphere MQ MQI client.

CipherSpecs and CipherSuites in IBM WebSphere MQ:

WebSphere MQ supports both SSL and TLS CipherSpecs, and RSA and Diffie-Hellman algorithms.

WebSphere MQ supports SSL V3 and TLS V1.0 and V1.2 CipherSpecs.

WebSphere MQ supports the RSA and Diffie-Hellman key exchange and authentication algorithms. The size of the key used during the 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 175
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 WebSphere MQ:

This topic provides information about how to configure WebSphere 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 WebSphere MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance. The IBM WebSphere MQ 7.5 readme file lists the version of Suite B enforced by each product maintenance level. If you configure WebSphere MQ to enforce Suite B compliance, always consult the readme file when planning to apply maintenance. The readme file is at http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg27006097.

On Windows, UNIX, and Linux systems, WebSphere MQ can be configured to conform to the Suite B compliant TLS 1.2 profile at the security levels shown in Table 1.

Table 7. 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 WebSphere MQ is not to comply with the Suite B standard, IBM WebSphere MQ can be configured to conform to either, or both security levels on Windows, UNIX and Linux systems. Following the successful configuration of IBM WebSphere 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 WebSphere MQ CipherSpecs, see "Specifying CipherSpecs" on page 376

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 WebSphere 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 282

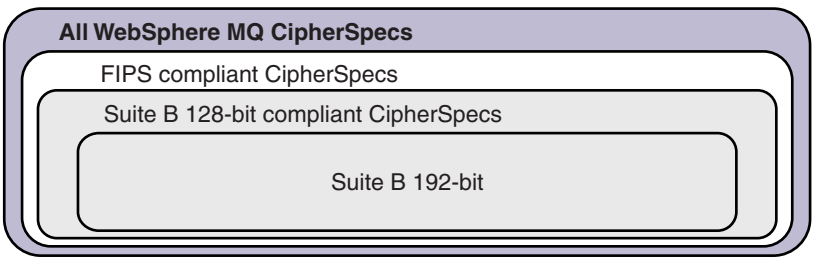
To request a CA-signed digital certificate for Suite B production use, see “Requesting a personal certificate on UNIX, Linux, and Windows systems” on page 284.

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 WebSphere MQ is configured for FIPS 140-2 compliant operation. It is therefore possible to configure WebSphere 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:



Configuring WebSphere MQ for Suite B compliant operation

For information about how to configure IBM WebSphere MQ on Windows, UNIX and Linux for Suite B compliant operation, see "Configuring WebSphere MQ for Suite B."

IBM WebSphere MQ does not support Suite B compliant operation on the IBM i and z/OS platforms. The WebSphere 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 269
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

Configuring WebSphere MQ for Suite B:

IBM WebSphere 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 WebSphere 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 177. For more information about Suite B configuration and its effect on SSL and TLS channels, see "NSA Suite B Cryptography in IBM WebSphere MQ" on page 188.

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

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 options below:

1. By setting the **EncryptionPolicySuiteB** field in the MQSCO structure on an MQCONNX call to one or more of the values below:
   - 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 values below:
   - 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 values below:
   - 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 WebSphere MQ classes for .NET, see MQEnvironment .NET class.

Certificate validation policies in WebSphere 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 WebSphere MQ” on page 192. 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 WebSphere 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 WebSphere MQ” on page 191.

Digital certificates and CipherSpec compatibility in IBM WebSphere 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 WebSphere MQ.

In previous releases of IBM WebSphere MQ, 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 v7.5 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 you need to display.

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 will produce 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 WebSphere MQ SSL and TLS connections, but are particularly relevant to users of Elliptic Curve cryptography.

The relationships between CipherSpecs and digital certificates are summarized in the following table:

Table 8. Relationships between CipherSpecs and digital certificates

Type	CipherSpec Name Prefix	Description	Required public key type	Digital signature encryption 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 only supported by WebSphere MQ queue managers and MQI clients on the UNIX, Linux, and Windows platforms.

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.

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 WebSphere 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.

A WebSphere MQ queue manager can only use a single personal certificate to identify itself. This means all channels on the queue manager will use the same digital certificate and therefore each queue manager may only use one type of CipherSpec at a time. Similarly, a WebSphere MQ client application can only use a single personal certificate to identify itself. This means that all SSL and TLS connections within a single application process will use the same digital certificate and therefore each client application process may only use one type of CipherSpec at a time.

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 QM1 must have a personal certificate with an Elliptic Curve key and an ECDSA-based digital signature. All clients and other queue managers which communicate directly with QM1 must therefore have digital certificates which satisfy the Type 1 CipherSpec requirements. Other channels connecting to queue manager QM1 may use other CipherSpecs (for example ECDHE_ECDSA_3DES_EDE_CBC_SHA256), but they may only use Type 1 CipherSpecs to communicate with QM1.

When planning your WebSphere MQ networks, consider carefully which channels require SSL or TLS and ensure that all of the clients and queue managers which need to interoperate use the same type of CipherSpecs and appropriate digital certificates. The IETF standards RFC 4492, RFC 5246 and RFC 6460 describe the detailed usage of Elliptic Curve CipherSpecs in TLS 1.2.

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.

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 WebSphere 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 WebSphere MQ" on page 188. 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 9. 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:

"Specifying CipherSpecs" on page 376
Specify 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 269
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

"NSA Suite B Cryptography in IBM WebSphere MQ" on page 188
This topic provides information about how to configure WebSphere 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 177
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.

CipherSpec values supported in WebSphere 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_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

Enabling deprecated 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 AMQ9788 in the error log for the queue manager.

It is possible for you to re-enable the deprecated CipherSpecs by editing the `qm.ini` file. Within the SSL stanza of the `qm.ini` file, add the following line:

```
SSL:
AllowWeakCipherSpec=Yes
```

You can also re-enable one or more of the deprecated CipherSpecs at runtime on the server by setting the environment variable *AMQ_SSL_WEAK_CIPHER_ENABLE* to any value. This environment variable enables the CipherSpecs regardless of the value that is specified in the `qm.ini` file.

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 IBM WebSphere 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.

Use of channel authentication records to control inbound channels is enabled using the **ALTER QMGR CHLAUTH(ENABLED)** switch.

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**.

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 WebSphere MQ system and must temporarily block the address before the firewall can be updated. These connection attempts might not even be coming from WebSphere MQ channels, but from other socket applications that are misconfigured to target your WebSphere 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 341 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 342 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 345 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 345 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 346 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 346 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 343 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 a WebSphere 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 344 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 344 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 343 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, queue manager name, or SSL or TLS DN, the most specific match is used. The match considered to be most specific is determined as follows.

- For a channel name:
  - The most specific match is a name without wildcards, for example A.B.C.
  - The most generic match is a single asterisk (*), which matches all channel names.
  - A pattern with an asterisk in the left-most position is more generic than a pattern with a defined value in the left-most position. Thus *.B.C is more generic than A.*.
  - A pattern with an asterisk in the second position is more generic than a pattern with a defined value in the second position, and similarly for each subsequent position. Thus A.*.C is more generic than A.B.*
  - Where two or more patterns have an asterisk in the same position, the one with fewer nodes following the asterisk is more generic. Thus A.* is more generic than A.*.C
- For an IP address:
  - The most specific match is a name without wildcards, for example 192.0.2.6.
  - The most generic match is a single asterisk (*), which matches all channel names.
  - A pattern with an asterisk in the left-most position is more generic than a pattern with a defined value in the left-most position. Thus *.0.2.6 is more generic than 192.*.
  - A pattern with an asterisk in the second position is more generic than a pattern with a defined value in the second position, and similarly for each subsequent position. Thus 192.*.2.6 is more generic than 192.0.*.

  - Where two or more patterns have an asterisk in the same position, the one with fewer nodes following the asterisk is more generic. Thus 192.* is more generic than 192.*.2.*.
  - 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 may 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.
  - A 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 a queue manager name:
  - The most specific match is a name without wildcards, for example 192.0.2.6.
  - The most generic match is a single asterisk (*), which matches all channel names.
  - A pattern with an asterisk in the left-most position is more generic than a pattern with a defined value in the left-most position. Thus *QUEUEMANAGER is more generic than QUEUEMANAGER*.
  - A pattern with an asterisk in the second position is more generic than a pattern with a defined value in the second position, and similarly for each subsequent position. Thus Q*MANAGER is more generic than QUEUE*.
  - Where two or more patterns have an asterisk in the same position, the one with fewer characters following the asterisk is more generic. Thus Q* is more generic than Q*MGR.
- For an SSL or TLS Distinguished Name (DN), the precedence order of substrings is as follows:

Table 10. 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

Table 10. Precedence order of substrings (continued)

Order	DN substring	Name
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, WebSphere 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 214
This section deals with remote messaging aspects of security.

Message security in WebSphere MQ

Message security in IBM WebSphere MQ infrastructure is provided by a separately licensed component WebSphere MQ Advanced Message Security.

WebSphere MQ Advanced Message Security (WebSphere MQ AMS) expands WebSphere 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, WebSphere MQ AMS verifies that a sender of message data is authorized to place signed messages on a target queue.

Related concepts:

"WebSphere MQ Advanced Message Security" on page 426
IBM WebSphere MQ Advanced Message Security (WebSphere MQ AMS) is a separately licensed component of WebSphere MQ that provides a high level of protection for sensitive data flowing through the WebSphere 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 a WebSphere MQ environment.

You can use WebSphere 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.

WebSphere 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 UNIX, Linux and Windows systems, if you ignore these aspects and do nothing, you cannot use WebSphere MQ.

Security considerations are described below.

Authority to administer WebSphere MQ

WebSphere MQ administrators need authority to:

- Issue commands to administer WebSphere MQ
- Use the WebSphere MQ Explorer

For more information, see:

- "Authority to administer WebSphere MQ on UNIX, Linux, and Windows systems" on page 355

Authority to work with WebSphere MQ objects

Applications can access the following WebSphere MQ objects by issuing MQI calls:

- Queue managers
- Queues
- Processes
- Namelists
- Topics

Applications can also use Programmable Command Format (PCF) commands to access these WebSphere MQ objects, and to access channels and authentication information objects as well. These objects can be protected by WebSphere MQ so that the user IDs associated with the applications need authority to access them.

For more information, see "Authorization for applications to use WebSphere MQ" on page 208.

Channel security

The user IDs associated with message channel agents (MCAs) need authority to access various WebSphere 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 229.

Additional considerations

You need to consider the following aspects of security only if you are using certain WebSphere MQ function or base product extensions:

- "Security for queue manager clusters" on page 238
- "Security for WebSphere MQ Publish/Subscribe" on page 239
- "Security for WebSphere MQ internet pass-thru" on page 241

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 WebSphere 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. WebSphere 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 WebSphere MQ. For more information see:

- "Mapping a remote queue manager to an MCAUSER user ID" on page 343
- "Mapping a client asserted user ID to an MCAUSER user ID" on page 344
- "Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID" on page 344
- "Mapping an IP address to an MCAUSER user ID" on page 346

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 a WebSphere 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.

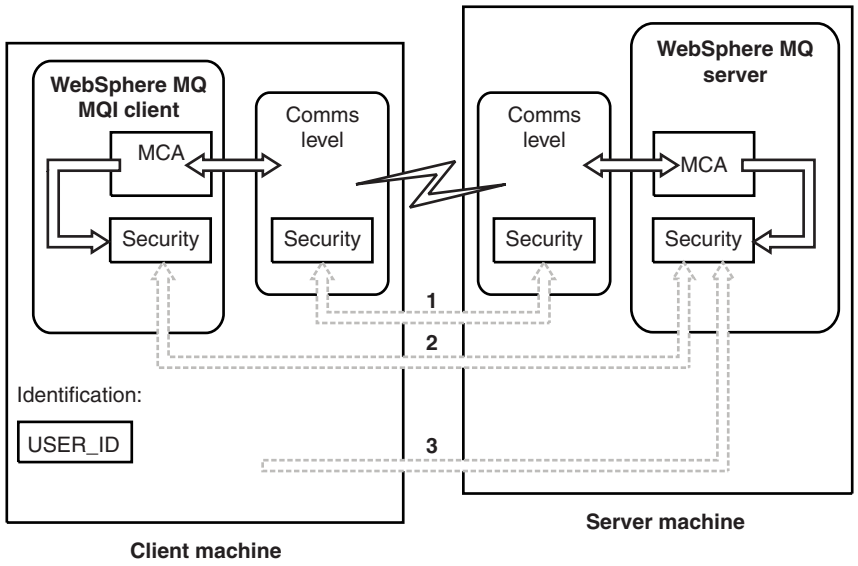


Figure 51. Security in a client/server connection

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 171
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 198.
3. 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.
4. 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 WebSphere 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 WebSphere 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 307. For more information about defining channel authentication records, see "Mapping a client asserted user ID to an MCAUSER user ID" on page 344.

   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 WebSphere MQ MQI client.

User IDs:

If the WebSphere MQ MQI client is on Windows and the WebSphere MQ server is also on Windows and has access to the domain on which the client user ID is defined, WebSphere MQ supports user IDs of up to 20 characters. On UNIX and Linux platforms and configurations, the maximum length is 12 characters.

A WebSphere 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`.

Note that `UNKNOWN` is a reserved user ID and the `NOBODY` user ID also have special meanings to WebSphere MQ. Creating user IDs in the operating system called `UNKNOWN` or `NOBODY` could have unintended results.

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.

Planning authorization

Plan the users who will have administrative authority and plan how to authorize users of applications to appropriately use IBM WebSphere MQ objects, including those connecting from a IBM WebSphere MQ MQI client.

Individuals or applications must be granted access in order to use IBM WebSphere MQ. What access they require depend on the roles they undertake and the tasks which they need to perform. Authorization in IBM WebSphere MQ can be subdivided into two main categories:

- Authorization to perform administrative operations
- Authorization for applications to use IBM WebSphere 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 WebSphere MQ

WebSphere MQ administrators need authority to perform various functions. This authority is obtained in different ways on different platforms.

WebSphere MQ administrators need authority to:

- Issue commands to administer WebSphere MQ
- Use the WebSphere MQ Explorer

For more information, see the topic appropriate to your operating system.

Authority to administer WebSphere MQ on UNIX and Windows systems:

A IBM WebSphere MQ administrator is a member of the mqm group. This group has access to all IBM WebSphere MQ resources and can issue IBM WebSphere MQ control commands. An administrator can grant specific authorities to other users.

To be a WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ resources.

All members of the mqm group have access to all WebSphere 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 WebSphere MQ resources.

Administrators can use the control command **runmqsc** to issue WebSphere 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 WebSphere MQ Explorer issues PCF commands to perform administration tasks. Administrators require no additional authorities to use the WebSphere MQ Explorer to administer a queue manager on the local system. When the WebSphere 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 WebSphere MQ."
- For commands that operate on channels, channel initiators, listeners, and clusters, see Channel security.

For more information about the authority you need to administer WebSphere MQ on UNIX and Windows systems, see the related information.

Authorization for applications to use WebSphere MQ

When applications access objects, the user IDs associated with the applications need appropriate authority.

Applications can access the following WebSphere MQ objects by issuing MQI calls:

- Queue managers
- Queues
- Processes
- Namelists
- Topics

Applications can also use PCF commands to administer WebSphere 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 WebSphere MQ for z/OS. The applications supplied with WebSphere MQ for z/OS include:

- The operations and control panels
- The WebSphere MQ utility program, CSQUTIL
- The dead letter queue handler utility, CSQUDLQH

Applications that use WebSphere MQ classes for Java, WebSphere MQ classes for JMS, WebSphere 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 WebSphere MQ objects. For more information about these user IDs and the authorities they require, see "Channel authorization" on page 229.

On z/OS, applications can also use MQSC commands to access these WebSphere MQ objects but command security and command resource security provide the authority checks in these circumstances.

On IBM i, a user that issues a CL command in Group 2 might require authority to access a WebSphere MQ object associated with the command. For more information, see "When authority checks are performed."

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 WebSphere 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 WebSphere MQ. WebSphere MQ assumes that users are signed on to the underlying operating system and are been authenticated by it.

When an application opens a WebSphere 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. Onz/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 a WebSphere 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 211.

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 WebSphere MQ objects on UNIX, Linux and Windows systems:

The authorization service component provided with IBM WebSphere MQ is called the *object authority manager (OAM)*. It provides access control via authentication and authorization checks.

1. Authentication.

   The authentication check performed by the OAM provided with IBM WebSphere 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 WebSphere MQ does not offer this. You must write a new authorization service component, or obtain one from a vendor.

2. 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 UNIX, Linux and Windows systems, the *authorization service* provides the access control when an application issues an MQI call to access a IBM WebSphere 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 WebSphere MQ objects, even when UAC is enabled.

Additionally, on Windows systems, the SYSTEM account has full access to IBM WebSphere MQ resources.

The authorization service also provides authority checks when a PCF command operates on one of these IBM WebSphere MQ objects or an authentication information object. The equivalent MQSC command encapsulated within an Escape PCF command is treated in the same way.

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 WebSphere MQ products.

The authorization service component provided with IBM WebSphere 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 WebSphere 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 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 WebSphere MQ continues to observe a 12 character restriction on all UNIX platforms. If you use a user ID of greater than 12 characters, IBM WebSphere 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 a IBM WebSphere 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.

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.

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 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 11. 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.

Security for remote messaging

This section deals with remote messaging aspects of security.

You must provide users with authority to use the IBM WebSphere 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 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 WebSphere 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 WebSphere 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 WebSphere MQ supplies an installable authorization service. You can choose to install an alternative service.

The authorization service component supplied with IBM WebSphere 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 WebSphere 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 12. 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 217 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 used 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 217 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 WebSphere 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.

Ensure that the MCAUSER field is restricted to 12 characters on Windows platforms because any extra characters will be truncated which might lead to authorization failures.

The IBM WebSphere 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 WebSphere 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 WebSphere 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 224
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" on page 231
SSL support in WebSphere 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 52 on page 219.

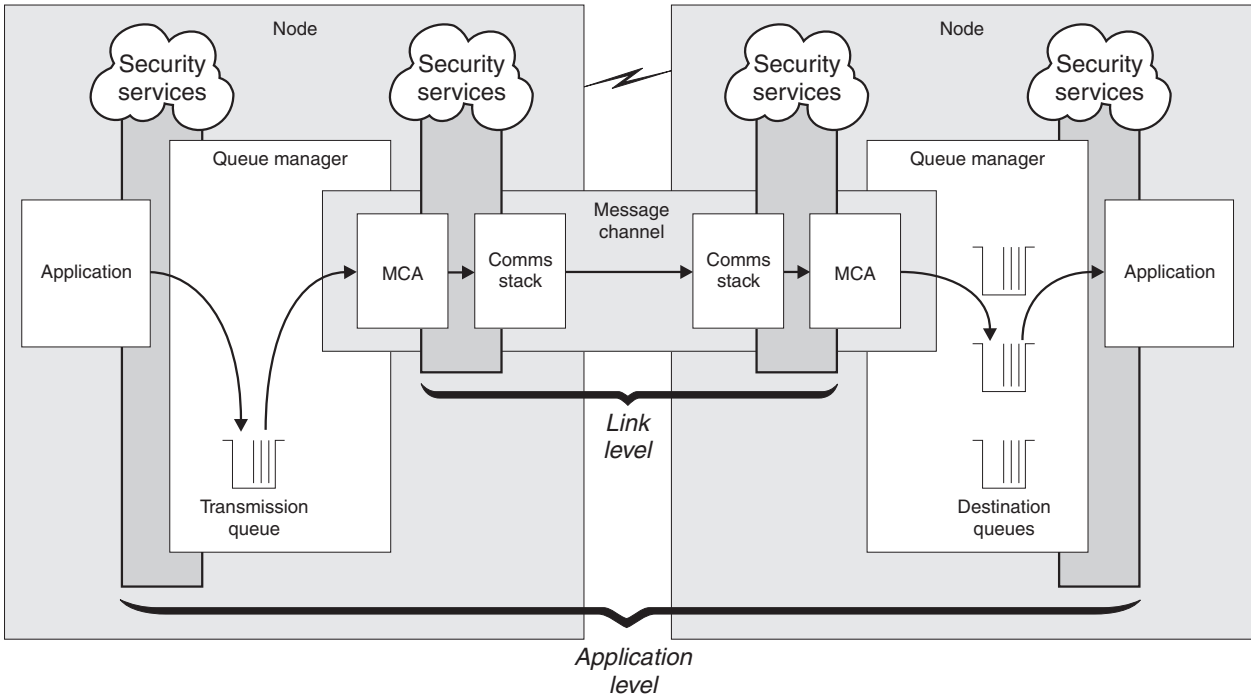


Figure 52. Link level security and application level security

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 52.

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 WebSphere 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 WebSphere 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 52 on page 219.

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 WebSphere MQ

The primary means of provision of confidentiality and data integrity in WebSphere MQ is by the use of SSL or TLS. For more information about the use of SSL and TLS in WebSphere MQ, see "WebSphere MQ support for SSL and TLS" on page 180. For authentication, WebSphere 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 198.

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 224. The same topic also describes the channel exit program that is supplied with IBM WebSphere 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 WebSphere MQ, or a combination of any of these working together. Application level security is illustrated in Figure 52 on page 219.

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.

WebSphere MQ Advanced Message Security:

WebSphere MQ Advanced Message Security (WebSphere MQ AMS) is a separately licensed component of WebSphere MQ that provides a high level of protection for sensitive data flowing through the WebSphere 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 WebSphere MQ. However, such solutions require specialist skills and can be complicated and expensive to maintain. WebSphere MQ Advanced Message Security helps address these challenges when moving information around the enterprise between virtually every type of commercial IT system.

WebSphere MQ Advanced Message Security extends the security features of WebSphere 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 WebSphere MQ servers and clients.
- It integrates with WebSphere MQ Managed File Transfer to provide an end-to-end secure messaging solution.

For more information, see “WebSphere MQ Advanced Message Security” on page 426.

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, WebSphere 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 WebSphere 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 WebSphere 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 53 on page 225 and are described in the following topics.

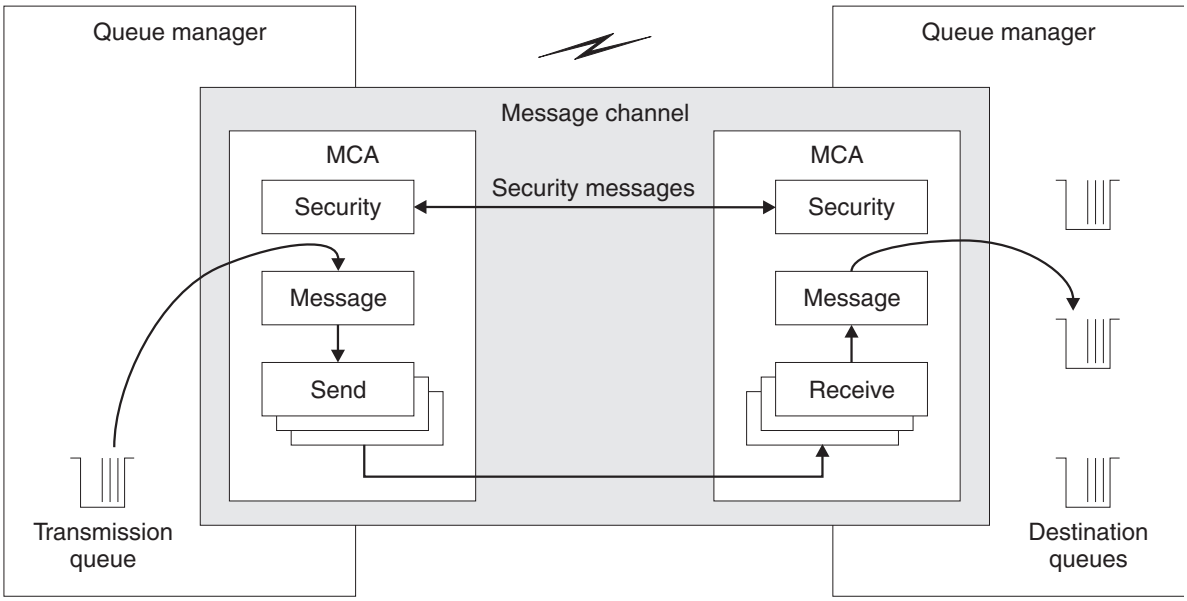


Figure 53. Security, message, send, and receive exits on a message channel

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 221.

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 WebSphere 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 222.

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 WebSphere 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 WebSphere 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 222.

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 might choose to use WebSphere MQ Advanced Message Security (AMS) to digitally sign messages in order to protect against unauthorized modification. You can also use API exit programs if standard facilities do not satisfy your requirements.

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" on page 231
SSL support in WebSphere 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 WebSphere MQ" on page 179
You can use a data integrity service to detect whether a message has been modified.

"WebSphere MQ Advanced Message Security" on page 223
WebSphere MQ Advanced Message Security (WebSphere MQ AMS) is a separately licensed component of WebSphere MQ that provides a high level of protection for sensitive data flowing through the WebSphere 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 WebSphere 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 WebSphere MQ using the IBM WebSphere MQ interfaces
Configure IBM WebSphere MQ to issue instrumentation events, specifically command events and configuration events.

Changes made to IBM WebSphere MQ outside its control
Some changes can affect how IBM WebSphere MQ behaves, but cannot be directly monitored by IBM WebSphere 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 WebSphere 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 WebSphere MQ can be configured to issue instrumentation events.

Application activity within IBM WebSphere MQ
To audit the actions of applications, for example opening of queues and putting and getting of messages, configure IBM WebSphere 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 WebSphere 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 IBM WebSphere 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 a user ID that has access to various WebSphere MQ resources.

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 WebSphere MQ, channels can be protected by SSL or 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 WebSphere 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.
  - If you use a user ID that lacks WebSphere administrative privileges, then you must grant dsp and ctrlx authority for the channel to that user ID for the channel to work. 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 198
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.

WebSphere MQ channel initiators are not WebSphere MQ objects; access to them is not controlled by the OAM. WebSphere 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 15 on page 251.

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 224 for more information about channel exits.

Protecting channels with SSL:

SSL support in WebSphere MQ uses the queue manager authentication information object, and various MQSC commands. You must also consider your use of digital certificates.

Commands and attributes for SSL support

The Secure Sockets Layer (SSL) protocol provides channel security, with protection against eavesdropping, tampering, and impersonation. WebSphere MQ support for SSL enables you to specify, on the channel definition, that a particular channel uses SSL 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:

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:

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 event message is reported if a channel using SSL fails to establish an SSL connection.

SSLFIPS
Specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in WebSphere 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 conversation before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

The following channel parameters support SSL:

SSLCAUTH
Defines whether WebSphere MQ requires and validates a certificate from the SSL client.

SSLCIPH
Specifies the encryption strength and function (CipherSpec), for example NULL_MD5 or RC4_MD5_US. 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, see

- "WebSphere MQ support for SSL and TLS" on page 180

For details of MQSC commands associated with SSL, see

- ALTER AUTHINFO
- DEFINE AUTHINFO
- DELETE AUTHINFO
- DISPLAY AUTHINFO

For details of PCF commands associated with SSL, see

- Change, Copy, and Create Authentication Information Object
- Delete Authentication Information Object
- Inquire Authentication Information Object

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 282 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 certificate. Similarly, if your organization's security policy requires that a particular CipherSpec be used, then you must obtain a suitable digital certificate.

For more information on the relationship between CipherSpecs and digital certificates, refer to "Digital certificates and CipherSpec compatibility in IBM WebSphere MQ" on page 192

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 WebSphere MQ, see "Certificate validation policies in WebSphere MQ" on page 191.

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, WebSphere 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.

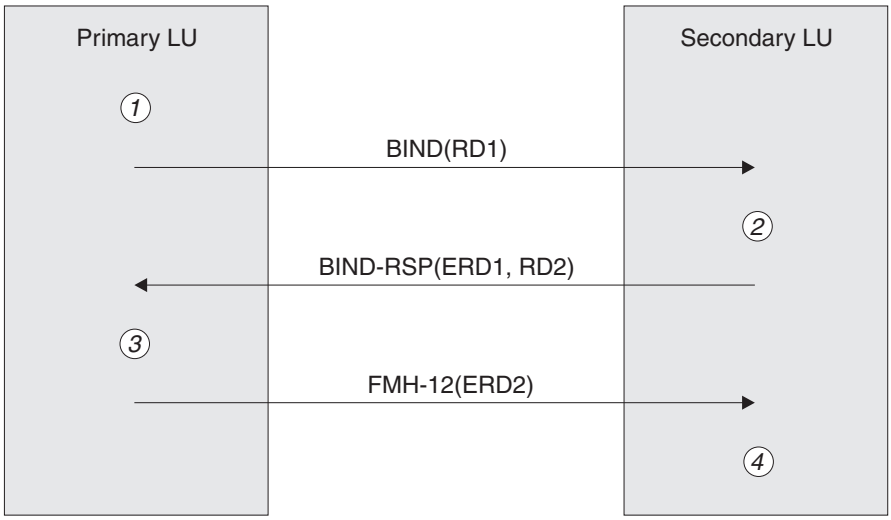
Session level authentication:

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 54 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 54. 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 54.

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 WebSphere MQ provides support for conversation level authentication.

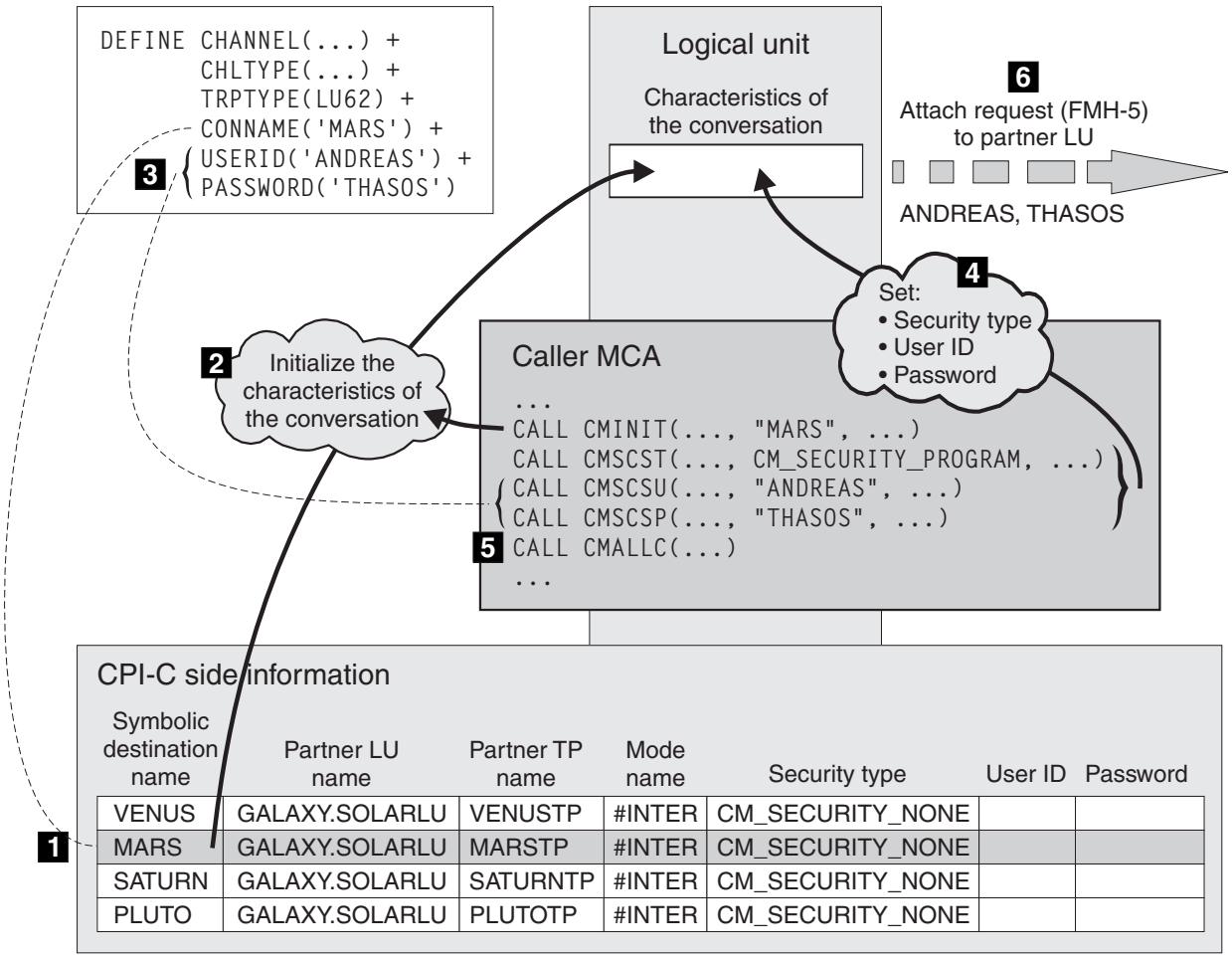
For more information about conversation level authentication, see *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808. For information specific to z/OS, see *z/OS MVS™ Planning: APPC/MVS Management*, SA22-7599.

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 WebSphere MQ on UNIX systems, and Windows systems:

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 WebSphere MQ for WebSphere MQ on UNIX systems, and WebSphere MQ for Windows is illustrated in Figure 55 on page 237. The numbers in the diagram correspond to the numbers in the description that follows.



Symbolic destination name	Partner LU name	Partner TP name	Mode name	Security type	User ID	Password
VENUS	GALAXY.SOLARLU	VENUSTP	#INTER	CM_SECURITY_NONE		
MARS	GALAXY.SOLARLU	MARSTP	#INTER	CM_SECURITY_NONE		
SATURN	GALAXY.SOLARLU	SATURNTP	#INTER	CM_SECURITY_NONE		
PLUTO	GALAXY.SOLARLU	PLUTOTP	#INTER	CM_SECURITY_NONE		

Figure 55. WebSphere 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.

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, WebSphere MQ decides which queue manager to send it to. When many applications send messages to the cluster queue, WebSphere 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, WebSphere 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.

Related tasks:

"Preventing queue managers receiving messages" on page 406
You can prevent a cluster queue manager from receiving messages it is unauthorized to receive by using exit programs.

Security for WebSphere MQ Publish/Subscribe

There are additional security considerations if you are using WebSphere MQ Publish/Subscribe.

In a publish/subscribe system, there are two types of application: publisher and subscriber. *Publishers* supply information in the form of WebSphere 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 WebSphere 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 WebSphere MQ Multicast.

WebSphere 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 WebSphere MQ applications (For example, LLM applications, see Multicast interoperability with WebSphere 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 WebSphere MQ is based on user IDs. For more information on this subject, see "Access control for clients" on page 216.

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 163.

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 WebSphere 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 341.

Security for WebSphere MQ internet pass-thru

Internet pass-thru can simplify communication through a firewall, but this has security implications.

WebSphere MQ internet pass-thru is a WebSphere MQ base product extension that is supplied in SupportPac MS81.

WebSphere MQ internet pass-thru enables two queue managers to exchange messages, or a WebSphere 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 WebSphere 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 WebSphere MQ system communicates with IPT, unless you are using SSLProxyMode in IPT, ensure that the CipherSpec used by WebSphere MQ matches the CipherSuite used by IPT:

- When IPT is acting as the SSL or TLS server and WebSphere MQ is connecting as the SSL or TLS client, the CipherSpec used by WebSphere 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 a WebSphere MQ SSL or TLS server, the IPT CipherSuite must match the CipherSpec defined on the receiving WebSphere MQ channel.

If you migrate from IPT to the integrated WebSphere MQ SSL and TLS support, transfer the digital certificates from IPT Using iKeyman.

For more information about WebSphere MQ internet pass-thru, see *MS81: WebSphere MQ internet pass-thru*, available from the following address: http://www.ibm.com/software/integration/support/supportpacs/

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.

WebSphere 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 WebSphere MQ
: You can define the set of users who can issue commands to administer WebSphere MQ.

Who can use WebSphere 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 WebSphere 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 WebSphere MQ; do not use standard operating system commands to grant or revoke authorizations to MQI resources.

Connecting to WebSphere 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 WebSphere MQ do no work when you use terminal services, try setting the **Create global objects** user right:

1. Open the Administrative Tools panel:

   Windows 2003 and Windows XP
   Access this panel using **Control Panel** > **Administrative Tools**.

   Windows Vista and Windows Server 2008
   Access this panel using **Control Panel** > **System and Maintenance** > **Administrative Tools**.
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 2003 and Windows XP
Access this panel using **Control Panel** > **Administrative Tools** > **Computer Management**.

Windows Vista and Windows Server 2008
Access this panel using **Control Panel** > **System and Maintenance** > **Administrative Tools** > **Computer Management**.

Windows 7
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 13, Table 14, Table 15 on page 251, and Table 16 on page 252 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 13. Security authorization needed for MQCONN calls

Authorization required for:	Queue object (1 on page 252)	Process object	Queue manager object
MQCONN	Not applicable	Not applicable	MQZAO_CONNECT

Table 14. Security authorization needed for MQOPEN calls

Authorization required for:	Queue object (1 on page 252)	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 252)**	MQZAO_INPUT	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) **(3 on page 252)**	MQZAO_OUTPUT	Not applicable	Not applicable

Table 14. Security authorization needed for MQOPEN calls (continued)

Authorization required for:	Queue object (1 on page 252)	Process object	Queue manager object
MQOO_PASS_ IDENTITY_CONTEXT **(4 on page 252)**	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ALL_ CONTEXT **(4 on page 252, 5 on page 252)**	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT **(4 on page 252, 5 on page 252)**	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(6 on page 252)**
MQOO_SET_ ALL_CONTEXT **(4 on page 252, 7 on page 252)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6 on page 252)**
MQOO_OUTPUT (Transmission queue) **(8 on page 252)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6 on page 252)**
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	**(9 on page 252)**	**(9 on page 252)**	MQZAO_ALTERNATE_ USER_AUTHORITY **(9 on page 252, 10 on page 252)**

Table 15. Security authorization needed for MQPUT1 calls

Authorization required for:	Queue object (1 on page 252)	Process object	Queue manager object
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_CONTEXT **(11 on page 252)**	Not applicable	No check
MQPMO_PASS_ALL _CONTEXT	MQZAO_PASS_ ALL_CONTEXT **(11 on page 252)**	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_CONTEXT **(11 on page 252)**	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT **(6 on page 252)**
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT **(11 on page 252)**	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6 on page 252)**
(Transmission queue) **(8 on page 252)**	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT **(6 on page 252)**
MQPMO_ALTERNATE_ USER_AUTHORITY	**(12 on page 252)**	Not applicable	MQZAO_ALTERNATE_ USER_AUTHORITY **(10 on page 252)**

Table 16. Security authorization needed for MQCLOSE calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQCO_DELETE	MQZAO_DELETE **(13)**	Not applicable	Not applicable
MQCO_DELETE _PURGE	MQZAO_DELETE **(13)**	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 257)**

Object	Authorization required
Queue	MQZAO_CREATE **(2 on page 257)**
Topic	MQZAO_CREATE **(2 on page 257)**
Process	MQZAO_CREATE **(2 on page 257)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(2 on page 257)**
Authentication information	MQZAO_CREATE **(2 on page 257)**
Channel	MQZAO_CREATE **(2 on page 257)**
Client connection channel	MQZAO_CREATE **(2 on page 257)**
Listener	MQZAO_CREATE **(2 on page 257)**
Service	MQZAO_CREATE **(2 on page 257)**
Communication information	MQZAO_CREATE **(2 on page 257)**

DEFINE *object* **REPLACE (1 on page 257, 3 on page 257)**

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 263)**

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	
Reset Cluster	'mqm' group membership required	
Suspend Queue Manager Cluster	'mqm' group membership required	
Resume Queue Manager Cluster	'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.

WebSphere 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 WebSphere MQ security when you are running WebSphere MQ in a Windows environment.

The SSPI channel exit program:

WebSphere 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 WebSphere MQ Version 7.0.

This service is typically used on an MQI channel to enable a server queue manager to authenticate a WebSphere 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 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 WebSphere 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 WebSphere 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 WebSphere MQ, because WebSphere 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 WebSphere 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 WebSphere MQ user IDs the required level of authority.

When you have problems with WebSphere 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 WebSphere MQ relies on group membership information to implement its security policy, it is important that the user ID that is performing WebSphere 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 WebSphere 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 WebSphere 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 a IBM WebSphere MQ Windows service to run under a domain user

Windows 2000 domain with non-default, or Windows 2003 and Windows Server 2008 domain with default, security permissions:

Installation of WebSphere MQ behaves differently on these operating systems depending on whether a local user or domain user performs the installation.

If a **local** user installs WebSphere MQ, the Prepare WebSphere MQ Wizard detects that the local user created for the IBM WebSphere MQ Windows service can retrieve the group membership information of the installing user. The Prepare WebSphere 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 WebSphere MQ Windows service needs to run under a domain user account with particular settings and authorities. The Prepare WebSphere 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 WebSphere MQ, the Prepare WebSphere MQ Wizard detects that the local user created for the IBM WebSphere MQ Windows service cannot retrieve the group membership information of the installing user. In this case, the Prepare WebSphere MQ Wizard always prompts the user for the account details of the domain user account for the IBM WebSphere MQ Windows service to use.

When the IBM WebSphere MQ Windows service needs to use a domain user account, WebSphere MQ cannot operate correctly until this has been configured using the Prepare WebSphere MQ Wizard. The Prepare WebSphere MQ Wizard does not allow the user to continue with other tasks, until the Windows service has been configured with a suitable account.

If a Windows 2000 domain has been configured with non-default security permissions, the usual solution to enable WebSphere MQ to work correctly is to configure it with a suitable domain user account, as described above.

See Creating and setting up domain accounts for WebSphere MQ for more information.

Configuring IBM WebSphere MQ Services to run under a domain user on Windows:

Use the Prepare IBM WebSphere 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 WebSphere MQ Service.

For more information see Changing the password of the IBM WebSphere MQ Windows service user account

Applying security template files to Windows:

Applying a template might affect the security settings applied to WebSphere MQ files and directories. If you use the highly secure template, apply it before installing WebSphere 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 WebSphere MQ files and directories. If you want to use the Highly Secure template, configure your machine before you install WebSphere MQ.

If you apply the highly secure template to a machine on which WebSphere MQ is already installed, all the permissions you have set on the WebSphere 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 WebSphere 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 2003 , Windows XP, Windows Vista, and Windows Server 2008 all 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 WebSphere MQ. The network APIs used by WebSphere MQ do not support all the configurations that are supported by the domain functional level. As a result, WebSphere 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 WebSphere MQ:

The account under which WebSphere 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 WebSphere MQ that are configured to run at a security level higher than usual.

WebSphere MQ requires SYNCHRONIZE access to application processes in order to coordinate certain actions. APAR IC35116 changed WebSphere MQ so that the appropriate privileges are specified. However, the account under which WebSphere 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 WebSphere MQ will modify the process to grant SYNCHRONIZE authority for WebSphere MQ administrators. To configure additional authority to the user ID under which WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ client application programs dynamically as required. Only the OpenSSL libraries that are provided by IBM WebSphere MQ are supported for use with IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ client for HP Integrity NonStop Server. The IBM WebSphere MQ client for HP Integrity NonStop Server is tested with the following entropy daemons:

- amqjkdm0 (as provided by the IBM WebSphere 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 WebSphere MQ MQI client security

You must consider WebSphere 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.

  You should use mutual authentication within SSL or TLS. For more information, see “Working with SSL or TLS” on page 272
- 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 318.

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 344.

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 198.

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 WebSphere 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. WebSphere MQ does not guarantee to reject all such connections and it is your responsibility to determine whether your WebSphere MQ configuration is FIPS-compliant.

Related concepts:

"Federal Information Processing Standards (FIPS) for UNIX, Linux, and Windows" on page 183
When cryptography is required on an SSL or TLS channel on Windows, UNIX and Linux systems, WebSphere 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 WebSphere MQ client to load a mixed set of libraries from two different GSKit V8.0 installations. Executing aIBM WebSphere MQ client application in a Db2 environment can cause this error.

To avoid this error, include the IBM WebSphere MQ library directories at the front of the library path so that the IBM WebSphere 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 WebSphere 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 UNIX, Linux or Windows systems

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 275.

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 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 WebSphere 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 WebSphere 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 WebSphere MQ client, ibmwebspheremq followed by your logon user ID changed to lower case, for example ibmwebspheremqmyuserid.

IBM WebSphere 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” on page 365.

Working with SSL or TLS

These topics give instructions for performing single tasks related to using SSL or TLS with IBM WebSphere MQ.

Many of them are used as steps in the higher-level tasks described in the following sections:

- “Identifying and authenticating users” on page 303
- “Authorizing access to objects” on page 318
- “Confidentiality of messages” on page 364
- “Data integrity of messages” on page 385
- “Keeping clusters secure” on page 402

Working with SSL or TLS on HP Integrity NonStop Server

Describes the IBM WebSphere MQ client for HP Integrity NonStop Server OpenSSL security implementation, including security services, components, supported protocol versions, supported CipherSpecs, and unsupported security functionality.

IBM WebSphere 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 WebSphere MQ client for HP Integrity NonStop Server comprises the following components:

- OpenSSL libraries and the **openssl** command.
- IBM WebSphere MQ password stash command, **amqrsslc**.

The following required components for SSL or TLS client channel operation are not provided with the IBM WebSphere 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 WebSphere 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 WebSphere 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
- FIPS_WITH_DES_CBC_SHA
- FIPS_WITH_3DES_EDE_CBC_SHA
- 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 WebSphere 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 WebSphere 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 WebSphere MQ client for HP Integrity NonStop Server uses the following, case sensitive, named files in the key repository location:
- "Personal certificate store" on page 274

- "Certificate trust store"
- "Pass phrase stash file"
- "Certificate revocation list file" on page 275

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 trust store:

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 WebSphere 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 WebSphere 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 UNIX, Linux and Windows systems

On UNIX, Linux and Windows systems, Secure Sockets Layer (SSL) support is installed with IBM WebSphere MQ.

For more detailed information about certificate validation policies, see Certificate validation and trust policy design.

Using iKeyman, iKeycmd, runmqakm, and runmqckm:

On UNIX, Linux and Windows systems, manage keys and digital certificates with the iKeyman GUI or from the command line using iKeycmd or runmqakm.

- For **UNIX and Linux** systems:
  - Use the **strmqikm** command to start the iKeyman GUI.
  - Use the **runmqckm** command to perform tasks with the iKeycmd command line interface.
  - Use the **runmqakm** 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 using the iKeyman user interface, or by using the **iKeycmd** or **runmqakm** commands.

About this task

An SSL or TLS connection requires a *key repository* at each end of the connection. Each IBM WebSphere MQ queue manager and IBM WebSphere MQ MQI client must have access to a key repository. For more information, see "The SSL or TLS key repository" on page 181.

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 WebSphere MQ MQI client. SSL and TLS use that certificate for authentication purposes. On UNIX, Linux, and Windows systems, IBM WebSphere MQ uses `ibmwebspheremq` as a label prefix to avoid confusion with certificates for other products. The prefix is followed by the name of the queue manager or IBM WebSphere MQ MQI client user logon ID, changed to lowercase. Ensure that you specify the entire certificate label in lowercase.

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 WebSphere 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 a WebSphere 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 280 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 a IBM WebSphere 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 WebSphere 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 278 or "Accessing and securing your key database files on UNIX and Linux systems" on page 279.

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 279.

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:\Program Files\IBM\WebSphere MQ\qmgrs\QM1\ssl` (on Windows systems).

       The path must match the value of the **SSLKeyRepository** attribute of the queue manager.
     - For a IBM WebSphere MQ client: `/var/mqm/ssl` (on UNIX and Linux systems) or `C:\mqm\ssl` (on Windows systems).
  7. Click **Open**. 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**.
  11. Click **OK**. The Personal Certificates window opens.
  12. Set the access permissions as described in "Accessing and securing your key database files on Windows" on page 278 or "Accessing and securing your key database files on UNIX and Linux systems" on page 279.
- 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 WebSphere MQ, it must be `cms`.)

-stash
Saves the key database password to a file.

-fips
Disables the use of the BSafe cryptographic library. Only the ICC component is used and this component must be successfully initialized 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 behaviour 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 behaviour 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 293.

Locating the key repository for a WebSphere 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 obtain the location of your WebSphere MQ MQI client's key database file. 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 a WebSphere MQ MQI client on UNIX, Linux or Windows systems." The value set in an MQCONNX call overrides the value of MQSSLKEYR.

Specifying the key repository location for a WebSphere MQ MQI client on UNIX, Linux or Windows systems:

There is no default key repository for a WebSphere 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 your WebSphere MQ MQI client's key database file in either of 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 WebSphere MQ SSL environment using the WebSphere 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 iKeyman, iKeycmd, or runmqakm.

Note: WebSphere 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” on page 366.

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. WebSphere MQ supports three different types of CipherSpec. For details, see “Interoperability of Elliptic Curve and RSA CipherSpecs” on page 193 in the “Digital certificates and CipherSpec compatibility in IBM WebSphere MQ” on page 192 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 WebSphere 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 **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 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, type:
   - For a queue manager, `ibmwebspheremq` followed by the name of your queue manager folded to lowercase. For example, for `QM1`, `ibmwebspheremqqm1`, or,
   - For a WebSphere MQ client, `ibmwebspheremq` followed by your logon user ID folded to lowercase, for example `ibmwebspheremqmyuserid` .
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 167.
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.
-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.
-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. This mode disables the use of the BSafe cryptographic library. Only the 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:

You can request a personal certificate by using the **strmqikm** (iKeyman) GUI, or from the command line using the **runmqckm** or **runmqakm** 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:

- WebSphere 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. WebSphere MQ supports three different types of CipherSpec. For details, see "Interoperability of Elliptic Curve and RSA CipherSpecs" on page 193 in the "Digital certificates and CipherSpec compatibility in IBM WebSphere MQ" on page 192 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 300.

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 following labels:
   - For a queue manager, enter ibmwebspheremq followed by the name of your queue manager changed to lowercase. For example, for a queue manager called QM1, enter ibmwebspheremqqm1.
   - For an IBM WebSphere MQ MQI client, enter ibmwebspheremq followed by your logon user ID, all in lowercase; for example, ibmwebspheremqmyuserid .
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 167.
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

Use the following commands to request a personal certificate by using either the **runmqckm** or **runmqakm** 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.

-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.

-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. This mode disables the use of the BSafe cryptographic library. Only the ICC component is used and this component must be successfully initialized 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.

Renewing an existing personal certificate on UNIX, Linux, and Windows systems:

You can renew a personal certificate by using the iKeyman user interface, or by using the **iKeycmd** or **runmqakm** commands.

Before you begin

If you have a requirement to use larger key sizes for your personal certificates, the renewal steps described below do not work, because the recreated certificate request is generated from an existing key.

Follow the steps described in "Requesting a personal certificate on UNIX, Linux, and Windows systems" on page 284 to create a new certificate request, using the key sizes you require. This process replaces your existing key.

About this task

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 a WebSphere 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 :

- 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` .

If you are using cryptographic hardware, refer to "Importing a personal certificate to your PKCS #11 hardware" on page 302.

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`.

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 **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.
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`.

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: If the certificate 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 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. You must add the certificates in strictly descending order starting from the root, followed by the CA certificate immediately below it in the chain.

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 **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 **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 :

- 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.
`-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 291).

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. This window allows the labels of certificates being imported to be changed 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. This can be used to change the personal certificate label to that required by WebSphere MQ in order to associate the certificate with the particular queue manager or client (`ibmwebspheremqqm1` for example).

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.

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 from a Microsoft .pfx file:

Folow this procedure to mport from a Microsoft .pfx file using iKeyman. You cannot use runmqakm to import a .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 in the form `ibmwebspheremq`, followed by the queue manager name or the WebSphere MQ MQI client user logon ID in lower case.
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

Importing 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 292).

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. This mode disables the use of the BSafe cryptographic library. Only the 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.

Generating strong passwords for key repository protection:

You can generate strong passwords for key repository protection using the **runmqakm** 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 WebSphere 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 a WebSphere 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 WebSphere 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 299. 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 a IBM WebSphere 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 on page 298, type a password in the **Password** field.
- If you selected **Create new secondary key database file** in step 8 on page 298:
  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 WebSphere MQ, it must be `cms`.)

  -stash
  Saves the key database password to a file.

  -fips
  Disables the use of the BSafe cryptographic library. Only the ICC component is used and this component must be successfully initialized 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 a IBM WebSphere MQ MQI client to request a personal certificate for your cryptographic hardware.

Using the iKeyman user interface:
About this task

Note: WebSphere 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 298.
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 following labels:
   - For a queue manager, enter `ibmwebspheremq` followed by the name of your queue manager changed to lowercase. For example, for a queue manager called `QM1`, enter `ibmwebspheremqqm1`.
   - For a IBM WebSphere MQ MQI client, enter `ibmwebspheremq` followed by your logon user ID, all in lowercase; for example, `ibmwebspheremqmyuserid` .
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 167.
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

Use the following commands to request a personal certificate by using either the **runmqckm** or **runmqakm** 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.

-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.

-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. This mode disables the use of the BSafe cryptographic library. Only the ICC component is used and this component must be successfully initialized 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.

Importing a personal certificate to your PKCS #11 hardware:

Use this procedure for either a queue manager or a IBM WebSphere 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 298.
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. Type a label. For example, you might use the same label as when you requested the personal certificate. Note that the label must be in the correct IBM WebSphere MQ format:
   - For a queue manager, `ibmwebspheremq` followed by the name of your queue manager in lowercase. For example, for a queue manager called QM1, the label would be: `ibmwebspheremqqm1`.
   - For a IBM WebSphere MQ MQI client, `ibmwebspheremq` followed by your logon user ID in lowercase. For example, for a user ID `MyUserID`, the label would be: `ibmwebspheremqmyuserid`.
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 This mode disables the use of the BSafe cryptographic library. Only the ICC component is used and this component must be successfully initialized 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.

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.

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 WebSphere MQ client application. In this situation, mutual authentication actually takes place between two queue managers, or between a queue manager and the user of a WebSphere 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 263.

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 171.

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 383. 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 235.

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 308.

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.

Privileged users

A privileged user is one that has full administrative authorities for WebSphere MQ.

In addition to the users listed in the following table, members of any group with `+crt` authority for queues are indirectly administrators. Similarly, any user that has `+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 17. Privileged users by platform.

A table of privileged users. On Windows, SYSTEM, all members of the mqm group, and all members of the Administrators group are privileged users. On UNIX and Linux systems, all members of the mqm group are privileged users. On IBM i, the profiles (users) qmqm and qmqmadm, all members of the qmqmadm group, and any user defined with the *ALLOBJ setting are privileged users.

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

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 WebSphere 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.

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 WebSphere 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 a WebSphere 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 263.

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 171.

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 383. 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 235.

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" on page 308.

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 UNIX, Linux and Windows systems, WebSphere 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. IBM WebSphere MQ classes for Java and IBM WebSphere MQ classes for JMS cannot use the OCSP information in a client channel definition table file. However, you can configure OCSP as described in the section Using Online Certificate Protocol.

On z/Os and IBM i WebSphere MQ SSL support checks for revoked certificates using CRLs and ARLs on LDAP servers only.

For more information about Certificate

Authorities, see “Digital certificates” on page 166.

Revoked certificates and OCSP

WebSphere 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 WebSphere MQ on Windows, UNIX and Linux systems.

To check the revocation status of a digital certificate using OCSP, WebSphere 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:

- WebSphere MQ cannot access the OCSP responder.
- The OCSP responder has sent a response, but WebSphere 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 WebSphere 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 WebSphere 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), WebSphere 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, WebSphere 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 WebSphere 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, WebSphere MQ cannot verify the digital signature on the OCSP response and the OCSP check results in an Unknown outcome, which might cause WebSphere 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, WebSphere 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.
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.

Property Name	Description
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

WebSphere MQ's 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.

WebSphere MQ maintains a cache of CRLs and ARLs that have been accessed in the preceding 12 hours.

When a queue manager or WebSphere MQ MQI client receives a certificate, it checks the CRL to confirm that the certificate is still valid. WebSphere MQ first checks in the cache, if there is a cache. If the CRL is not in the cache, WebSphere 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 WebSphere MQ finds an available CRL. If the namelist is not specified, or is specified with a blank value, CRLs are not checked.

For more information about LDAP, see Using lightweight directory access protocol services with WebSphere MQ for Windows.

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 WebSphere 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 56 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 56. Sample LDIF file for a Certificate Authority. This might vary from implementation to implementation.

Figure 57 on page 314 shows the DIT structure that your LDAP server creates when you load the sample LDIF file shown in Figure 56 together with a similar file for CA2, an imaginary Certificate Authority set up by the PKI organization, also within IBM.

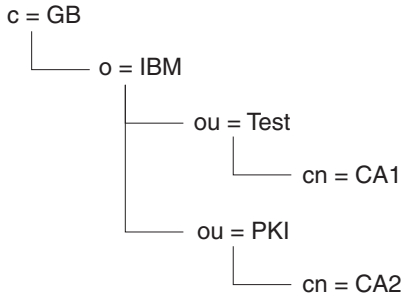


Figure 57. Example of an LDAP Directory Information Tree structure

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.

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.

   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.

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.
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.

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 using WebSphere MQ Explorer:

You can use WebSphere 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 a WebSphere MQ MQI client:

You have three options for specifying the LDAP servers that hold CRLs for checking by a WebSphere 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 a WebSphere MQ MQI client channel running on Linux (zSeries platform).

Location of an OCSP responder, and of LDAP servers that hold CRLs:

On a WebSphere 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.

When a WebSphere 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 a WebSphere 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 a WebSphere 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 a WebSphere 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 the server platforms AIX, HP-UX, Linux, 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, a WebSphere 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 WebSphere 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.

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 WebSphere MQ classes for Java and WebSphere MQ classes for JMS:

WebSphere MQ classes for Java and WebSphere MQ classes for JMS access CRLs differently from other platforms.

For information about working with CRLs and ARLs with WebSphere MQ classes for Java, see Using certificate revocation lists

For information about working with CRLs and ARLs with WebSphere 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 WebSphere MQ Explorer.

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 WebSphere MQ and to work with IBM WebSphere 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 WebSphere MQ objects.

You must be suitably authorized to use these commands, as described in “Authority to administer WebSphere MQ on UNIX, Linux, and Windows systems” on page 355. User IDs that are authorized to administer WebSphere 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 a WebSphere MQ object on UNIX or Linux systems and Windows

Use the **setmqaut** control command, or the **MQCMD_SET_AUTH_REC** PCF command to give users, and groups of users, access to WebSphere MQ objects.

For a full definition of the **setmqaut** control command and its syntax, see setmqaut, and 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 or Linux systems and Windows" on page 320.
- 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 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.

Using OAM generic profiles on UNIX or Linux systems and Windows

OAM generic profiles enable you to set the authority a user has to many objects at once, rather than having to issue separate **setmqaut** commands against each individual object when it is created.

Using generic profiles in the **setmqaut** command 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 the example above, 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, and 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 WebSphere 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 the example above, 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:

Use the **dmpmqaut** control command or the **MQCMD_INQUIRE_AUTH_RECS** PCF command to dump the current authorizations associated with a specified profile.

For a full definition of the **dmpmqaut** control command and its syntax, see dmpmqaut, and 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 WebSphere 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, or the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command to view the authorizations that a specific principal or group has for a particular object.

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, and for a full definition of the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command and its syntax, see Inquire Entity Authority.

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 a WebSphere MQ object

To change the level of access that a user or group has to an object, use the **setmqaut** command. To revoke the access of a particular user that is a member of a group that has authorization, remove the user from the group.

The process of removing the user from a group is described in:

- "Creating and managing groups on Windows" on page 243
- "Creating and managing groups on HP-UX" on page 244
- "Creating and managing groups on AIX" on page 246
- "Creating and managing groups on Solaris" on page 247
- "Creating and managing groups on Linux" on page 248

.

The user ID that creates a WebSphere 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, and for a full definition of the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command and its syntax, see Inquire Entity Authority.

On Windows, delete the OAM entries corresponding to a particular Windows user account before deleting the user profile. It is 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 more information about the implications of setting the MQSNOAUT variable.
- 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 WebSphere 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 WebSphere 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 WebSphere 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."
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 332.
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 337.
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 338.
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 339
   b. Yes: See "Allowing user applications to connect to your queue manager" on page 339.

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 18. 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 328
Topics	Grant partial administrative access to the required topics, as described in "Granting limited administrative access to some topics" on page 329
Channels	Grant partial administrative access to the required channels, as described in "Granting limited administrative access to some channels" on page 329

Table 18. Granting partial administrative access to a subset of queue manager resources (continued)

The users need to administer objects of this type	Perform this action
The queue manager	Grant partial administrative access to the queue manager, as described in "Granting limited administrative access to a queue manager" on page 330
Processes	Grant partial administrative access to the required processes, as described in "Granting limited administrative access to some processes" on page 330
Namelists	Grant partial administrative access to the required namelists, as described in "Granting limited administrative access to some namelists" on page 331
Services	Grant partial administrative access to the required services, as described in "Granting limited administrative access to some services" on page 331

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.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  `setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName ReqdAction`
- 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.

  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.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  `setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName ReqdAction`
- 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. +ctrl. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

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.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  `setmqaut -m QMgrName -n ObjectProfile -t channel -g GroupName ReqdAction`
- 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.

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.

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.

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.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  `setmqaut -m QMgrName -n ObjectProfile -t process -g GroupName ReqdAction`
- 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.

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.

Procedure

- For UNIX, Linux and Windows systems, issue the following command:
  `setmqaut -m QMgrName -n ObjectProfile -t namelist -g GroupName ReqdAction`
- 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.

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.

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.

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 19. 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" on page 333
Topics	Grant full administrative access to the required topics, as described in "Granting full administrative access to some topics" on page 333
Channels	Grant full administrative access to the required channels, as described in "Granting full administrative access to some channels" on page 334

Table 19. Granting full administrative access to a subset of queue manager resources (continued)

The users need to administer objects of this type	Perform this action
The queue manager	Grant full administrative access to the queue manager, as described in "Granting full administrative access to a queue manager" on page 334
Processes	Grant full administrative access to the required processes, as described in "Granting full administrative access to some processes" on page 335
Namelists	Grant full administrative access to the required namelists, as described in "Granting full administrative access to some namelists" on page 336
Services	Grant full administrative access to the required services, as described in "Granting full administrative access to some services" on page 336

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.

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.

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.

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.

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.

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.

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.

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.

Procedure

- Using the wizard:
  1. In the WebSphere 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.

Procedure

- Using the wizard:
  1. In the WebSphere 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.

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 347.

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 340.

In both cases, set up connection security as described in “Setting up connection security” on page 347

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 347
You have applications that make use of topics	See “Controlling user access to topics” on page 352.
You have applications that inquire on the queue manager object	See “Granting authority to inquire on a queue manager” on page 353.
You have applications that use process objects	See “Granting authority to access processes” on page 354
You have applications that make use of namelists	See “Granting authority to access namelists” on page 354

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 341
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 305. 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 commands shown below, 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
) 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 WebSphere MQ rules for SSLPEER values. See WebSphere MQ rules for SSLPEER values.

user is the user ID to be used for all connections using the specified DN.

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 command shown below, 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 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') 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 WebSphere MQ rules for SSLPEER values. See WebSphere MQ rules for SSLPEER values.

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.

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.

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”
The application passes context	See “Granting authority to pass context” on page 349
The application puts messages on a clustered queue	See “Authorizing putting messages on remote cluster queues” on page 403
The application puts messages on a local queue	See “Granting authority to put messages to a local queue” on page 350
The application puts messages on a model queue	See “Granting authority to put messages to a model queue” on page 350
The application puts messages on a remote queue	See “Granting authority to put messages to a remote cluster queue” on page 351

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.

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(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 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.

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
    ```

- 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.

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.

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.

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 350. 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, and Windows systems, you can also use 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  QMgrNameObjectProfile UACC(NONE)
  PERMIT QMgrNameObjectProfile CLASS(MQADMIN)
  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 20. 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” on page 353

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.

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.

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.

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.

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.

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 WebSphere MQ on UNIX, Linux, and Windows systems

WebSphere MQ administrators can use all WebSphere 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.

WebSphere MQ administrators have authority to use all WebSphere MQ commands (including the commands to grant WebSphere MQ authorities for other users)

To be a WebSphere MQ administrator, you must be a member of a special group called the *mqm* group (or a member of the Administrators group on Windows systems). The mqm group is created automatically when WebSphere 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 WebSphere MQ. One of these control commands is **setmqaut**, which is used to grant authorities to other users to enable them to access or control WebSphere 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 can use the control command **runmqsc** to issue WebSphere 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 WebSphere MQ Explorer issues PCF commands to perform administration tasks. Administrators require no additional authorities to use the WebSphere MQ Explorer to administer a queue manager on the local

system. When the WebSphere 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 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 WebSphere 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

Additionally, on Windows systems, the SYSTEM account has full access to WebSphere 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 WebSphere 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 WebSphere 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 360.

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 WebSphere 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 do the following:

- 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 230).
- 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 WebSphere 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 WebSphere 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, WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ to use. This is described in the Configuring WebSphere MQ accounts.

Authority to work with WebSphere MQ objects on UNIX, Linux and Windows systems

All objects are protected by WebSphere 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 WebSphere 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 WebSphere MQ on UNIX, Linux, and Windows systems" on page 355
WebSphere MQ administrators can use all WebSphere 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. WebSphere 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 WebSphere MQ; WebSphere 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)
WebSphere 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).

WebSphere 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 252.

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 WebSphere MQ on UNIX, Linux and Windows systems

WebSphere MQ uses the security services provided by the underlying operating system, using the object authority manager. WebSphere MQ supplies commands to create and maintain access control lists.

An access control interface called the Authorization Service Interface is part of WebSphere MQ. WebSphere 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 326). 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 WebSphere 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 318 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.

WebSphere 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 318.

WebSphere 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. WebSphere MQ follows the decision of the OAM; if the OAM cannot make a decision, WebSphere 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. WebSphere 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.

WebSphere 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 362. Applications cannot alter this information unless they have been authorized to change context information.

Principals and groups:

Principals can belong to groups. You can grant access to a particular resource to groups rather than to individuals, to reduce the amount of administration required. On UNIX and Linux systems all Access Control Lists (ACLs) are based on groups, but on Windows systems, ACLS are based on user IDs and groups.

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 is 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

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 the PCF equivalent).

UNIX and Linux systems

All ACLs are based on groups. When a user is granted access to a particular resource, the primary group of the user ID 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 WebSphere 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 except that individual user IDs can be displayed in the ACL as well. You can have different users on different domains with the same user ID. WebSphere 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\GroupName
```

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 alternate 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 WebSphere 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 of the same name who is not in the local mqm group.

Windows security identifiers (SIDs):

WebSphere MQ on Windows uses the SID where it is available. If a Windows SID is not supplied with an authorization request, WebSphere 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 WebSphere MQ for Windows, WebSphere MQ stores the SID in the message descriptor. When WebSphere 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, WebSphere 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 WebSphere MQ authority might be granted. To prevent this problem, include an SID in each authorization request; the SID is used by WebSphere 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 a WebSphere MQ object. This is called *alternate-user authority*, and you can use it on any WebSphere 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 a WebSphere 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. *MCAUserIdentifier* contains the first 12 bytes of the MCA user identifier. If the MCA user identifier is not blank, it specifies the user identifier to be used by the message channel agent for authorization to access MQ resources. Ensure that the MCAUSER is less than 12 characters on Windows platform.

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 WebSphere 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 WebSphere 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 WebSphere MQ v7.1 may require updating. For more information see Channel security exit programs.

WebSphere MQ object authority manager user authentication

On WebSphere 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 WebSphere 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 307.

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 WebSphere 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.

Confidentiality of messages

To maintain confidentiality, encrypt your messages. There are various methods of encrypting messages in WebSphere 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 WebSphere 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 WebSphere MQ Advanced Message Security, see “WebSphere MQ Advanced Message Security” on page 223.The use of SSL and TLS with WebSphere MQ is described at “WebSphere MQ support for SSL and TLS” on page 180. The use of exit programs in message encryption is described at “Implementing confidentiality in user exit programs” on page 383.

Connecting two queue managers using SSL or TLS

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 obtain and manage your digital certificates. On a test system, you can use self-signed certificates or certificates issued by a local certificate authority (CA). On a production system, do not use self-signed certificates. For more information, see ../zs14140_.dita.

For full information about creating and managing certificates, see “Working with SSL or TLS on UNIX, Linux and Windows systems” on page 275.

This collection of topics introduces 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 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 WebSphere MQ implementation, the SSL or TLS server always requests a certificate from the client.

Notes:

1. In this context, an SSL client refers to the connection initiating the handshake.
2. See the Glossary for further details.

On UNIX, Linux and Windows systems, the SSL or TLS client sends a certificate only if it has one labeled in the correct WebSphere MQ format, which is `ibmwebspheremq` followed by the name of your queue manager changed to lowercase. For example, for `QM1`, `ibmwebspheremqqm1`.

WebSphere 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 lowercase.

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 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. For more information about connecting a queue manager anonymously, that is, when the SSL or TLS client does not send a certificate, see “Connecting two queue managers using one-way authentication” on page 370.

Using self-signed certificates for mutual authentication of two queue managers

Follow these sample instructions to implement mutual authentication between two queue managers, using self-signed SSL or TLS certificates.

About this task

Scenario:

- You have two queue managers, QM1 and QM2, which need to communicate securely. You require mutual authentication to be carried out between QM1 and QM2.
- You have decided to test your secure communication using self-signed certificates.

The resulting configuration looks like this:

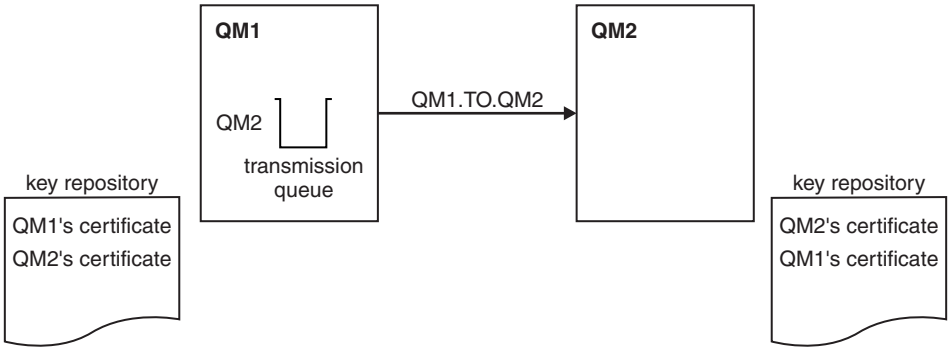


Figure 58. Configuration resulting from this task

In Figure 58, the key repository for QM1 contains the certificate for QM1 and the public certificate from QM2. The key repository for QM2 contains the certificate for QM2 and the public certificate from QM1.

Procedure

1. Prepare the key repository on each queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Create a self-signed certificate for each queue manager:
   - On UNIX, Linux, and Windows systems.
3. Extract a copy of each certificate:
   - On UNIX, Linux, and Windows systems.
4. Transfer the public part of the QM1 certificate to the QM2 system and vice versa, using a utility such as FTP.
5. Add the partner certificate to the key repository for each queue manager:
   - On UNIX, Linux, and Windows systems.
6. On QM1, define a sender channel and associated transmission queue, by issuing commands like the following example:

   ```
   DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM1.MACH.COM) XMITQ(QM2)
   SSLCIPH(RC4_MD5_US) DESCR('Sender channel using SSL from QM1 to QM2')

   DEFINE QLOCAL(QM2) USAGE(XMITQ)
   ```

   This example uses CipherSpec RC4_MD5. The CipherSpecs at each end of the channel must be the same.

7. On QM2, define a receiver channel, by issuing a command like the following example:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(RC4_MD5_US)
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 step 6, and use the same CipherSpec.
8. Start the channel.

Results

Key repositories and channels are created as illustrated in Figure 58 on page 366

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following examples.

From queue manager QM1, enter the following command:

```
DISPLAY CHS(QM1.TO.QM2) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
     4 : DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(QM1.TO.QM2)                     CHLTYPE(SDR)
   CONNAME(9.20.25.40)                     CURRENT
   RQMNAME(QM2)
   SSLCERTI("CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5E:02,CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(MQGET)
   XMITQ(QM2)
```

From queue manager QM2, enter the following command:

```
DISPLAY CHS(QM1.TO.QM2) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(QM2.TO.QM1)                     CHLTYPE(RCVR)
   CONNAME(9.20.35.92)                     CURRENT
   RQMNAME(QM1)
   SSLCERTI("CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
   XMITQ( )
```

In each case, the value of SSLPEER must match that of the DN in the partner certificate that was created in Step 2. The issuers name matches the peer name because the certificate is self-signed .

SSLPEER is optional. If it is specified, its value must be set so that the DN in the partner certificate (created in step 2) is allowed. For more information about the use of SSLPEER, see WebSphere MQ rules for SSLPEER values.

Using CA-signed certificates for mutual authentication of two queue managers

Follow these sample instructions to implement mutual authentication between two queue managers, using CA-signed SSL or TLS certificates.

About this task

Scenario:

- You have two queue managers called QMA and QMB, which need to communicate securely. You require mutual authentication to be carried out between QMA and QMB.
- In the future you are planning to use this network in a production environment, and therefore you have decided to use CA-signed certificates from the beginning.

The resulting configuration looks like this:

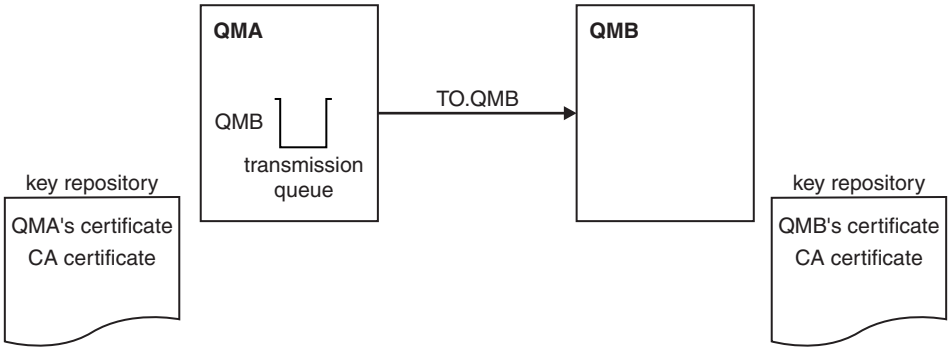


Figure 59. Configuration resulting from this task

In Figure 59, the key repository for QMA contains QMA's certificate and the CA certificate. The key repository for QMB contains QMB's certificate and the CA certificate. In this example both QMA's certificate and QMB's certificate were issued by the same CA. If QMA's certificate and QMB's certificate were issued by different CAs then the key repositories for QMA and QMB must contain both CA certificates.

Procedure

1. Prepare the key repository on each queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Request a CA-signed certificate for each queue manager. You might use different CAs for the two queue managers.
   - On UNIX, Linux, and Windows systems.
3. Add the Certificate Authority certificate to the key repository for each queue manager: If the Queue managers are using different Certificate Authorities then the CA certificate for each Certificate Authority must be added to both key repositories.
   - On UNIX, Linux, and Windows systems.
4. Add the CA-signed certificate to the key repository for each queue manager:
   - On UNIX, Linux, and Windows systems.
5. On QMA, define a sender channel and associated transmission queue by issuing commands like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QMB.MACH.COM) XMITQ(QMB)
SSLCIPH(RC2_MD5_EXPORT) DESCR('Sender channel using SSL from QMA to QMB')

DEFINE QLOCAL(QMB) USAGE(XMITQ)
```

This example uses CipherSpec RC4_MD5. The CipherSpecs at each end of the channel must be the same.

6. On QMB, define a receiver channel by issuing a command like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(RC2_MD5_EXPORT)
SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL to QMB')
```

The channel must have the same name as the sender channel you defined in step 6, and use the same CipherSpec.

7. Start the channel:

Results

Key repositories and channels are created as illustrated in Figure 59 on page 368.

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is like that shown in the following examples.

From queue manager QMA, enter the following command:

```
DISPLAY CHS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
     4 : DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QMB)                         CHLTYPE(SDR)
   CONNAME(9.20.25.40)                     CURRENT
   RQMNAME(QMB)
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMB,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(MQGET)
   XMITQ(QMB)
```

From the queue manager QMB, enter the following command:

```
DISPLAY CHS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QMB)                         CHLTYPE(RCVR)
   CONNAME(9.20.35.92)                     CURRENT
   RQMNAME(QMA)
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMA,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
   XMITQ( )
```

In each case, the value of SSLPEER must match that of the Distinguished Name (DN) in the partner certificate that was created in Step 2. The issuer name matches the subject DN of the CA certificate that signed the personal certificate added in Step 4.

Connecting two queue managers using one-way authentication

Follow these sample instructions to modify a system with mutual authentication to allow a queue manager to connect using one-way authentication to another; that is, when the SSL or TLS client does not send a certificate.

About this task

Scenario:

- Your two queue managers (QM1 and QM2) have been set up as in "Using CA-signed certificates for mutual authentication of two queue managers" on page 368.
- You want to change QM1 so that it connects using one-way authentication to QM2.

The resulting configuration looks like this:

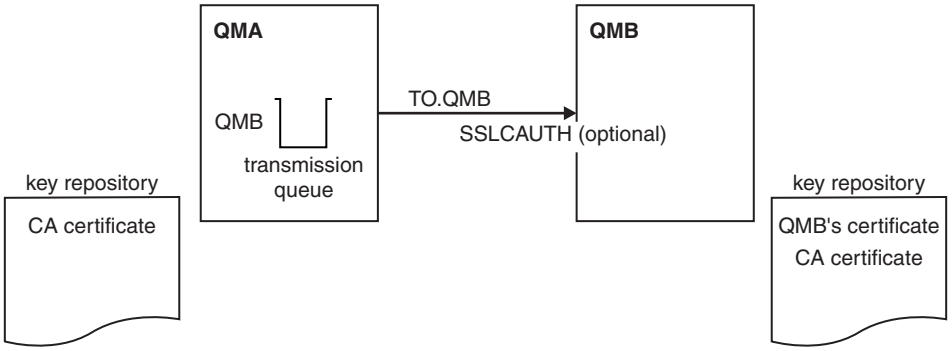


Figure 60. Queue managers allowing one-way authentication

Procedure

1. Remove QM1's personal certificate from its key repository, according to operating system:
   - On UNIX, Linux, and Windows systems. The certificate is labeled as follows:
     - `ibmwebspheremq` followed by the name of your queue manager folded to lower case. For example, for `QM1`, `ibmwebspheremqqm1`.
2. Optional: On QM1, if any SSL or TLS channels have run previously, refresh the SSL or TLS environment.
3. Allow anonymous connections on the receiver.

Results

Key repositories and channels are changed as illustrated in Figure 60

What to do next

If the sender channel was running and you issued the REFRESH SECURITY TYPE(SSL) command (in step 2), the channel restarts automatically. If the sender channel was not running, start it.

At the server end of the channel, the presence of the peer name parameter value on the channel status display indicates that a client certificate has flowed.

Verify that the task has been completed successfully by issuing some DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following examples:

From the QM1 queue manager, enter the following command:

```
DISPLAY CHS(TO.QM2) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
     4 : DISPLAY CHSTATUS(TO.QMB) SSLPEER
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QM2)                         CHLTYPE(SDR)
   CONNAME(9.20.25.40)                     CURRENT
   RQMNAME(QM2)
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMB,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(MQGET)
   XMITQ(QM2)
```

From the QM2 queue manager, enter the following command:

```
DISPLAY CHS(TO.QM2) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(TO.QM2) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QM2)                         CHLTYPE(RCVR)
   CONNAME(9.20.35.92)                     CURRENT
   RQMNAME(QMA)                            SSLCERTI( )
   SSLPEER( )                              STATUS(RUNNING)
   SUBSTATE(RECEIVE)                       XMITQ( )
```

On QM2, the SSLPEER field is empty, showing that QM1 did not send a certificate. On QM1, the value of SSLPEER matches that of the DN in QM2's personal certificate.

Connecting a client to a queue manager securely

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 obtain and manage your digital certificates. On a test system, you can use self-signed certificates or certificates issued by a local certificate authority (CA). On a production system, do not use self-signed certificates. For more information, see ../zs14140_.dita.

For full information about creating and managing certificates, see “Working with SSL or TLS on UNIX, Linux and Windows systems” on page 275.

This collection of topics introduces 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 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 WebSphere 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 WebSphere MQ format, which is `ibmwebspheremq` followed by your logon user ID changed to lowercase, for example `ibmwebspheremqmyuserid`.

WebSphere 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 lowercase.

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 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. For more information about connecting a queue manager anonymously, see “Connecting a client to a queue manager anonymously” on page 375.

Using self-signed certificates for mutual authentication of a client and queue manager

Follow these sample instructions to implement mutual authentication between a client and a queue manager, by using self-signed SSL or TLS certificates.

About this task

Scenario:

- You have a client, C1, and a queue manager, QM1, which need to communicate securely. You require mutual authentication to be carried out between C1 and QM1.
- You have decided to test your secure communication by using self-signed certificates.

DCM on IBM i does not support self-signed certificates, so this task is not applicable on IBM i systems.

The resulting configuration looks like this:

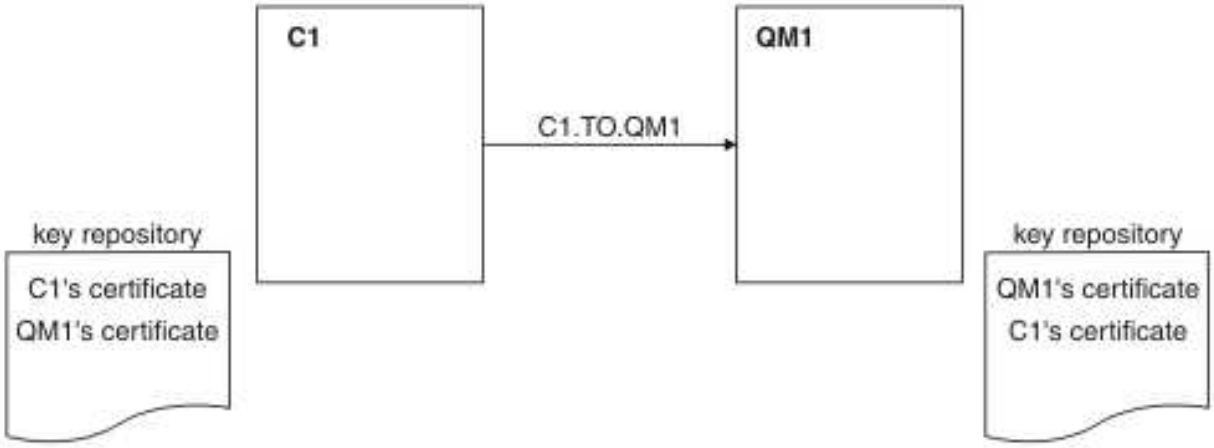


Figure 61. Configuration resulting from this task

In Figure 61, the key repository for QM1 contains the certificate for QM1 and the public certificate from C1. The key repository for C1 contains the certificate for C1 and the public certificate from QM1.

Procedure

1. Prepare the key repository on the client and queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Create self-signed certificates for the client and queue manager:
   - On UNIX, Linux, and Windows systems.
3. Extract a copy of each certificate:
   - On UNIX, Linux, and Windows systems.
4. Transfer the public part of the C1 certificate to the QM1 system and vice versa, using a utility such as FTP.

5. Add the partner certificate to the key repository for the client and queue manager:
   - On UNIX, Linux, and Windows systems.
6. Define a client-connection channel in either of the following ways:
   - Using the MQCONNX call with the MQSCO structure on C1, as described in Creating a client-connection channel on the WebSphere MQ MQI client.
   - Using a client channel definition table, as described in Creating server-connection and client-connection definitions on the server.
7. On QM1, define a server-connection channel, by issuing a command like the following example:

   ```
   DEFINE CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCIPH(RC4_MD5_US)
   SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL from C1 to QM1')
   ```

   The channel must have the same name as the client-connection channel you defined in step 6, and use the same CipherSpec.

Results

Key repositories and channels are created as illustrated in Figure 61 on page 372

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following example.

From queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(C1.TO.QM1)                    CHLTYPE(SVRCONN)
   CONNAME(9.20.35.92)                     CURRENT
   SSLCERTI("CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5E:02,CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

It is optional to set the SSLPEER filter attribute of the channel definitions. If the channel definition SSLPEER is set, its value must match the subject DN in the partner certificate that was created in Step 2. After a successful connection, the SSLPEER field in the DISPLAY CHSTATUS output shows the subject DN of the remote client certificate.

Using CA-signed certificates for mutual authentication of a client and queue manager

Follow these sample instructions to implement mutual authentication between a client and a queue manager, by using CA-signed SSL or TLS certificates.

About this task

Scenario:

- You have a client, C1, and a queue manager, QM1, which need to communicate securely. You require mutual authentication to be carried out between C1 and QM1.
- In the future you are planning to use this network in a production environment, and therefore you have decided to use CA-signed certificates from the beginning.

The resulting configuration looks like this:

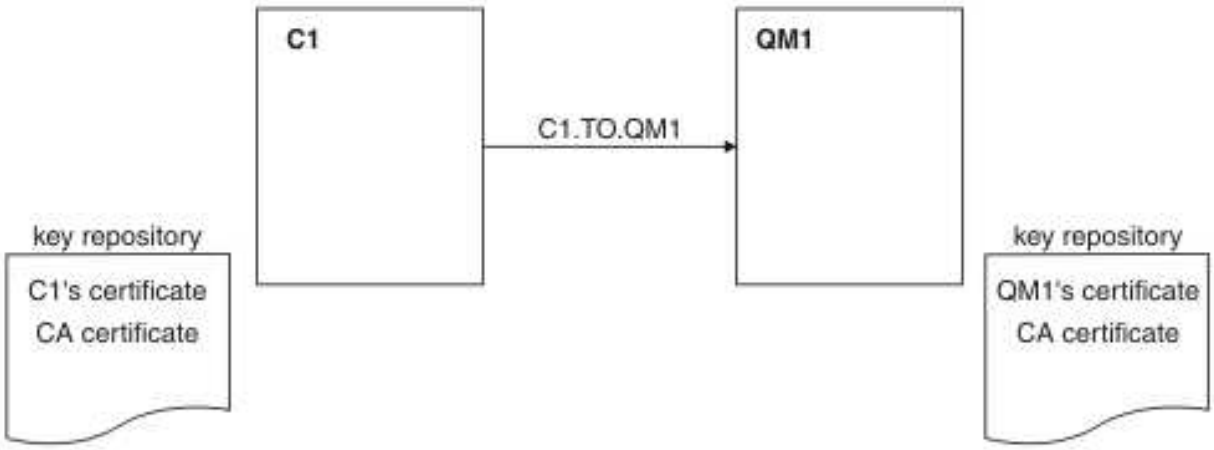


Figure 62. Configuration resulting from this task

In Figure 62, the key repository for C1 contains certificate for C1 and the CA certificate. The key repository for QM1 contains the certificate for QM1 and the CA certificate. In this example both C1's certificate and QM1's certificate were issued by the same CA. If C1's certificate and QM1's certificate were issued by different CAs then the key repositories for C1 and QM1 must contain both CA certificates.

Procedure

1. Prepare the key repository on the client and queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Request a CA-signed certificate for the client and queue manager. You might use different CAs for the client and queue manager.
   - On UNIX, Linux, and Windows systems.
3. Add the certificate authority certificate to the key repository for the client and queue manager. If the client and queue manager are using different Certificate Authorities then the CA certificate for each Certificate Authority must be added to both key repositories.
   - On UNIX, Linux, and Windows systems.
4. Add the CA-signed certificate to the key repository for the client and queue manager:
   - On UNIX, Linux, and Windows systems.
5. Define a client-connection channel in either of the following ways:
   - Using the MQCONNX call with the MQSCO structure on C1, as described in Creating a client-connection channel on the WebSphere MQ MQI client.
   - Using a client channel definition table, as described in Creating server-connection and client-connection definitions on the server.
6. On QM1, define a server-connection channel by issuing a command like the following example:

   ```
   DEFINE CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCIPH(RC2_MD5_EXPORT)
   SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL from C1 to QM1')
   ```

   The channel must have the same name as the client-connection channel you defined in step 6, and use the same CipherSpec.

Results

Key repositories and channels are created as illustrated in Figure 62.

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is like that shown in the following example.

From the queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(C1.TO.QM1)                      CHLTYPE(SVRCONN)
   CONNAME(9.20.35.92)                     CURRENT
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMA,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

The SSLPEER field in the DISPLAY CHSTATUS output shows the subject DN of the remote client certificate that was created in Step 2. The issuer name matches the subject DN of the CA certificate that signed the personal certificate added in Step 4.

Connecting a client to a queue manager anonymously

Follow these sample instructions to modify a system with mutual authentication to allow a queue manager to connect anonymously to another.

About this task

Scenario:

- Your queue manager and client (QM1 and C1) have been set up as in "Using CA-signed certificates for mutual authentication of a client and queue manager" on page 373.
- You want to change C1 so that it connects anonymously to QM1.

The resulting configuration looks like this:

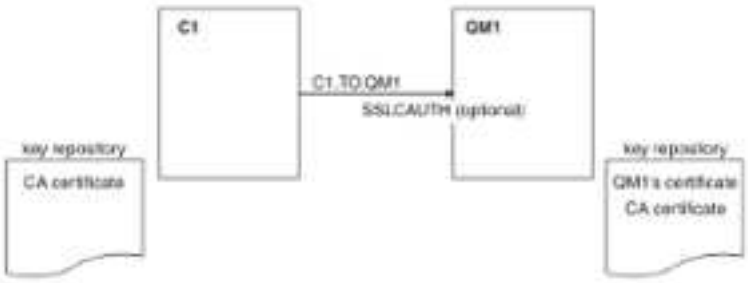


Figure 63. Client and queue manager allowing anonymous connection

Procedure

1. Remove the personal certificate from key repository for C1, according to operating system:
   - On UNIX, Linux, and Windows systems. The certificate is labeled as follows:
     - `ibmwebspheremq` followed by your logon user ID folded to lower case, for example `ibmwebspheremqmyuserid`.
2. Restart the client application, or cause the client application to close and reopen all SSL or TLS connections.

3. Allow anonymous connections on the queue manager, by issuing the following command:
   ```
   ALTER CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) SSLCAUTH(OPTIONAL)
   ```

Results

Key repositories and channels are changed as illustrated in Figure 63 on page 375

What to do next

At the server end of the channel, the presence of the peer name parameter value on the channel status display indicates that a client certificate has flowed.

Verify that the task has been completed successfully by issuing some DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following example:

From queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(C1.TO.QM1)                      CHLTYPE(SVRCONN)
   CONNAME(9.20.35.92)                     CURRENT
   SSLCERTI( )                             SSLPEER( )
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

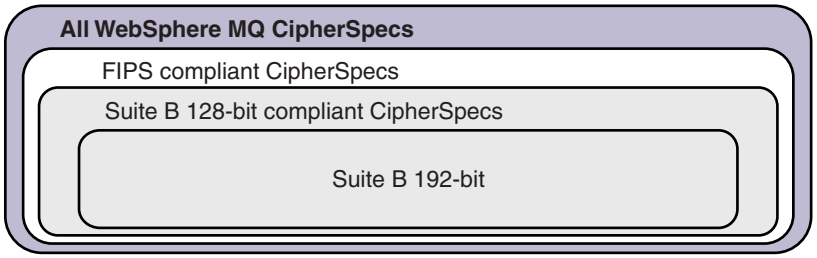
The SSLCERTI and SSLPEER fields are empty, showing that C1 did not send a certificate.

Specifying CipherSpecs

Specify 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 WebSphere MQ are FIPS compliant. Others, such as NULL_MD5, are not. Similarly, some of the FIPS compliant CipherSpecs are also Suite B compliant although others, 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:



Cipher specifications that you can use with IBM WebSphere MQ SSL and TLS support 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.

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
NULL_MD5 [a]	SSL 3.0	MD5	None	0	No	No	No
NULL_SHA [a]	SSL 3.0	SHA-1	None	0	No	No	No
RC4_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC4	40	No	No	No
RC4_MD5_US [a]	SSL 3.0	MD5	RC4	128	No	No	No
RC4_SHA_US [a]	SSL 3.0	SHA-1	RC4	128	No	No	No
RC2_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC2	40	No	No	No
DES_SHA_EXPORT [2] [a]	SSL 3.0	SHA-1	DES	56	No	No	No
RC4_56_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	RC4	56	No	No	No
DES_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	DES	56	No	No	No
TLS_RSA_WITH_AES_128_CBC_SHA [a]	TLS 1.0	SHA-1	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA [4] [a]	TLS 1.0	SHA-1	AES	256	Yes	No	No
TLS_RSA_WITH_DES_CBC_SHA [a]	TLS 1.0	SHA-1	DES	56	No[5]	No	No
FIPS_WITH_DES_CBC_SHA [b]	SSL 3.0	SHA-1	DES	56	No[6]	No	No
TLS_RSA_WITH_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	256	Yes	No	No
ECDHE_ECDSA_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No
ECDHE_RSA_RC4_128_SHA256 [b]	TLS 1.2	SHA_1	RC4	128	No	No	No
ECDHE_ECDSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_ECDSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_RSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_RSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_ECDSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	Yes	No
ECDHE_ECDSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	Yes
ECDHE_RSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
ECDHE_RSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_NULL_SHA256 [b]	TLS 1.2	SHA-256	None	0	No	No	No
ECDHE_RSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
ECDHE_ECDSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
TLS_RSA_WITH_NULL_NULL [b]	TLS 1.2	None	None	0	No	No	No
TLS_RSA_WITH_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No

Notes:

1. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
2. 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.
3. The handshake key size is 1024 bits.
4. This CipherSpec cannot be used to secure a connection from the WebSphere MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
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. 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.

Platform support:

- a Available on all supported platforms.
- b Available only on UNIX, Linux, and Windows platforms.

Related concepts:

"Digital certificates and CipherSpec compatibility in IBM WebSphere MQ" on page 192
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 WebSphere MQ.

Related information:

DEFINE CHANNEL

ALTER CHANNEL

Deprecated CipherSpecs

A list of deprecated CipherSpecs that you are able to use with WebSphere MQ if necessary.

See "CipherSpec values supported in WebSphere MQ" on page 197 for more information on how you can enable deprecated CipherSpecs.

Deprecated CipherSpecs that you can use with WebSphere MQ TLS 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
All	DES_SHA_EXPORT[3]	SSL 3.0	SHA-1	DES	56	No	No	7.5.0.6
Linux Windows UNIX	DES_SHA_EXPORT1024[4]	SSL 3.0	SHA-1	DES	56	No	No	7.5.0.6
Linux Windows UNIX	FIPS_WITH_DES_CBC_SHA	SSL 3.0	SHA-1	DES	56	No[6]	No	7.5.0.6
Linux Windows UNIX	FIPS_WITH_3DES_EDE_CBC_SHA	SSL 3.0	SHA-1	3DES	168	No[7]	No	7.5.0.8
All	NULL_MD5	SSL 3.0	MD5	None	0	No	No	7.5.0.6
All	NULL_SHA	SSL 3.0	SHA-1	None	0	No	No	7.5.0.6
All	RC2_MD5_EXPORT[3]	SSL 3.0	MD5	RC2	40	No	No	7.5.0.7
All	RC4_MD5_EXPORT[3]	SSL 3.0	MD5	RC4	40	No	No	7.5.0.7
All	RC4_MD5_US	SSL 3.0	MD5	RC4	128	No	No	7.5.0.7
All	RC4_SHA_US	SSL 3.0	SHA-1	RC4	128	No	No	7.5.0.7
Linux Windows UNIX	RC4_56_SHA_EXPORT1024[4]	SSL 3.0	SHA-1	RC4	56	No	No	7.5.0.7
All	TRIPLE_DES_SHA_US	SSL 3.0	SHA-1	3DES	168	No	No	7.5.0.8
All	TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	SHA-1	DES	56	No[5]	No	7.5.0.6
Linux Windows UNIX	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	SHA-1	None	0	No	No	7.5.0.6
Linux Windows UNIX	ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	SHA-1	RC4	128	No	No	7.5.0.7
Linux Windows UNIX	ECDHE_RSA_NULL_SHA256	TLS 1.2	SHA-1	None	0	No	No	7.5.0.6
Linux Windows UNIX	ECDHE_RSA_RC4_128_SHA256	TLS 1.2	SHA-1	RC4	128	No	No	7.5.0.7
Linux Windows UNIX	TLS_RSA_WITH_NULL_NULL	TLS 1.2	None	None	0	No	No	7.5.0.6
All	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	SHA-256	None	0	No	No	7.5.0.6

Platform support[1]	CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[2]	Suite B	Update when deprecated
Linux Windows UNIX	TLS_RSA_WITH_RC4_128_SHA256	TLS 1.2	SHA-1	RC4	128	No	No	7.5.0.7
All	TLS_RSA_WITH_3DES_EDE_CBC_SHA[8]	TLS 1.0	SHA-1	3DES	168	Yes	No	7.5.0.8
Linux Windows UNIX	ECDHE_ECDSA_3DES_EDE_CBC_SHA256[8]	TLS 1.2	SHA-1	3DES	168	Yes	No	7.5.0.8
Linux Windows UNIX	ECDHE_RSA_3DES_EDE_CBC_SHA256[8]	TLS 1.2	SHA-1	3DES	168	Yes	No	7.5.0.8

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. 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.

Obtaining information about CipherSpecs using WebSphere MQ Explorer

You can use WebSphere MQ Explorer to display descriptions of CipherSpecs.

Use the following procedure to obtain information about the CipherSpecs in “Specifying CipherSpecs” on page 376:

1. Open **WebSphere 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 WebSphere 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 "Specifying CipherSpecs" on page 376. 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* at http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp
- For V5R4, the *IBM i product documentation* at http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp

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 two-character string representing a hexadecimal value. The hexadecimal codes correspond to the values defined in the SSL protocol.

For more information, refer to the description of gsk_environment_open() in the API reference chapter of *z/OS Cryptographic Services System SSL Programming*, SC24-5901, where there is a list of all the supported SSL V3.0 and TLS V1.0 cipher specifications in the form of 2-digit hexadecimal codes.

Considerations for WebSphere MQ clusters

With WebSphere MQ clusters it is safest to use the CipherSpec names in "Specifying CipherSpecs" on page 376. 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 406.

Specifying a CipherSpec for a WebSphere MQ MQI client

You have three options for specifying a CipherSpec for a WebSphere 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 WebSphere MQ classes for Java and WebSphere MQ classes for JMS

WebSphere MQ classes for Java and WebSphere MQ classes for JMS specify CipherSuites differently from other platforms.

For information about specifying a CipherSuite with WebSphere MQ classes for Java, see Secure Sockets Layer (SSL) support

For information about specifying a CipherSuite with WebSphere MQ classes for JMS, see Using Secure Sockets Layer (SSL) with WebSphere MQ classes for JMS

Resetting SSL and TLS secret keys

IBM WebSphere 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.

Queue manager

For a queue manager, use the command **ALTER QMGR** with the parameter **SSLRKEYC** to set the values used during key renegotiation. 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for .NET, see Getting and setting attribute values.

XMS .NET

For XMS .NET unmanaged clients, see Secure connections to an IBM WebSphere MQ queue manager.

Related information:

ALTER QMGR

DISPLAY QMGR

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" on page 383.

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 WebSphere 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,

Connecting two queue managers using SSL or TLS

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 obtain and manage your digital certificates. On a test system, you can use self-signed certificates or certificates issued by a local certificate authority (CA). On a production system, do not use self-signed certificates. For more information, see ../zs14140_.dita.

For full information about creating and managing certificates, see “Working with SSL or TLS on UNIX, Linux and Windows systems” on page 275.

This collection of topics introduces 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 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 WebSphere MQ implementation, the SSL or TLS server always requests a certificate from the client.

Notes:

1. In this context, an SSL client refers to the connection initiating the handshake.
2. See the Glossary for further details.

On UNIX, Linux and Windows systems, the SSL or TLS client sends a certificate only if it has one labeled in the correct WebSphere MQ format, which is `ibmwebspheremq` followed by the name of your queue manager changed to lowercase. For example, for `QM1`, `ibmwebspheremqqm1`.

WebSphere 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 lowercase.

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 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. For more information about connecting a queue manager anonymously, that is, when the SSL or TLS client does not send a certificate, see “Connecting two queue managers using one-way authentication” on page 370.

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.

About this task

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 the *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. However, you can change the label associated with a digital certificate if necessary.

How is the certificate label used?

IBM WebSphere 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.

Certificate labels follow a naming convention; you need to ensure that you use the correct label naming convention corresponding to the platform that you are using.

In this context, an SSL or TLS client refers to the connection partner initiating the handshake, which might be a IBM WebSphere 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 WebSphere MQ implementation, the SSL or TLS server always requests a certificate from the client and the client always provide a certificate to the server if a 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.

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" on page 370.

, UNIX, Linux, and Windows systems:
About this task

On , UNIX, Linux, and Windows systems, the SSL or TLS server sends a certificate to the client, only if the server finds one labeled in the correct IBM WebSphere MQ format. On these systems, the correct format is `ibmwebspheremq`, followed by the name of your queue manager changed to lower case.

For example, for a queue manager named `QM1`, the certificate label requirement is:

```
ibmwebspheremqqm1
```

If no certificate is found in the queue manager's key repository, matching the required label in the correct case and format, an error occurs and the SSL or TLS handshake fails.

WebSphere MQ client:
About this task

When connecting from a IBM WebSphere MQ client application, the SSL or TLS client sends a certificate only if it has one a certificate with a label in the format `ibmwebspheremq`, followed by the username of the user running the client application process.

For example, for the username `wasadmin`, the certificate label requirement is as shown, folded to lower case:

```
ibmwebspheremqwasadmin
```

The above label requirement applies to Message Service Clients for C, or C++, and .NET.

IBM WebSphere MQ Java or IBM WebSphere MQ JMS client:
About this task

IBM WebSphere MQ Java or IBM WebSphere 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 therefore are not 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 WebSphere 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 use `-Djavax.net.debug=ssl` on the command line, or set the variable within the application, or through configuration.

Related concepts:

"Importing a personal certificate into a key repository on UNIX, Linux or Windows systems" on page 293
Follow this procedure to import a personal certificate

Using self-signed certificates for mutual authentication of two queue managers

Follow these sample instructions to implement mutual authentication between two queue managers, using self-signed SSL or TLS certificates.

About this task

Scenario:

- You have two queue managers, QM1 and QM2, which need to communicate securely. You require mutual authentication to be carried out between QM1 and QM2.
- You have decided to test your secure communication using self-signed certificates.

The resulting configuration looks like this:

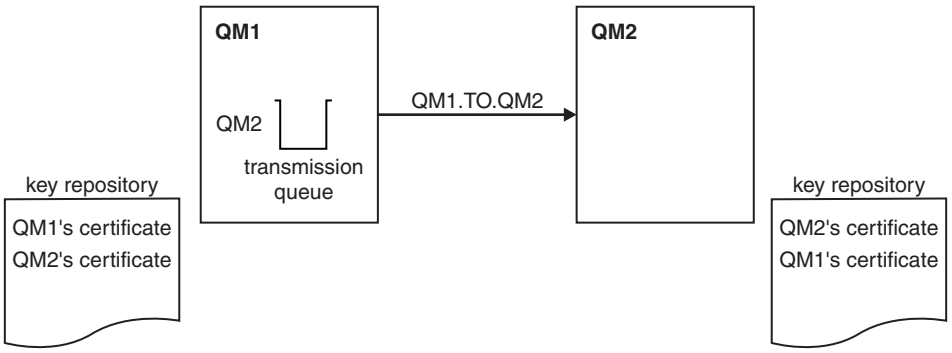


Figure 64. Configuration resulting from this task

In Figure 64, the key repository for QM1 contains the certificate for QM1 and the public certificate from QM2. The key repository for QM2 contains the certificate for QM2 and the public certificate from QM1.

Procedure

1. Prepare the key repository on each queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Create a self-signed certificate for each queue manager:
   - On UNIX, Linux, and Windows systems.
3. Extract a copy of each certificate:
   - On UNIX, Linux, and Windows systems.
4. Transfer the public part of the QM1 certificate to the QM2 system and vice versa, using a utility such as FTP.
5. Add the partner certificate to the key repository for each queue manager:
   - On UNIX, Linux, and Windows systems.

6. On QM1, define a sender channel and associated transmission queue, by issuing commands like the following example:

   ```
   DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM1.MACH.COM) XMITQ(QM2)
   SSLCIPH(RC4_MD5_US) DESCR('Sender channel using SSL from QM1 to QM2')

   DEFINE QLOCAL(QM2) USAGE(XMITQ)
   ```

   This example uses CipherSpec RC4_MD5. The CipherSpecs at each end of the channel must be the same.
7. On QM2, define a receiver channel, by issuing a command like the following example:

   ```
   DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(RC4_MD5_US)
   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 step 6, and use the same CipherSpec.
8. Start the channel.

Results

Key repositories and channels are created as illustrated in Figure 64 on page 388

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following examples.

From queue manager QM1, enter the following command:

```
DISPLAY CHS(QM1.TO.QM2) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
     4 : DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(QM1.TO.QM2)                     CHLTYPE(SDR)
   CONNAME(9.20.25.40)                     CURRENT
   RQMNAME(QM2)
   SSLCERTI("CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5E:02,CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(MQGET)
   XMITQ(QM2)
```

From queue manager QM2, enter the following command:

```
DISPLAY CHS(QM1.TO.QM2) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(QM2.TO.QM1)                     CHLTYPE(RCVR)
   CONNAME(9.20.35.92)                     CURRENT
   RQMNAME(QM1)
   SSLCERTI("CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
   XMITQ( )
```

In each case, the value of SSLPEER must match that of the DN in the partner certificate that was created in Step 2. The issuers name matches the peer name because the certificate is self-signed .

SSLPEER is optional. If it is specified, its value must be set so that the DN in the partner certificate (created in step 2) is allowed. For more information about the use of SSLPEER, see WebSphere MQ rules for SSLPEER values.

Using CA-signed certificates for mutual authentication of two queue managers

Follow these sample instructions to implement mutual authentication between two queue managers, using CA-signed SSL or TLS certificates.

About this task

Scenario:

- You have two queue managers called QMA and QMB, which need to communicate securely. You require mutual authentication to be carried out between QMA and QMB.
- In the future you are planning to use this network in a production environment, and therefore you have decided to use CA-signed certificates from the beginning.

The resulting configuration looks like this:

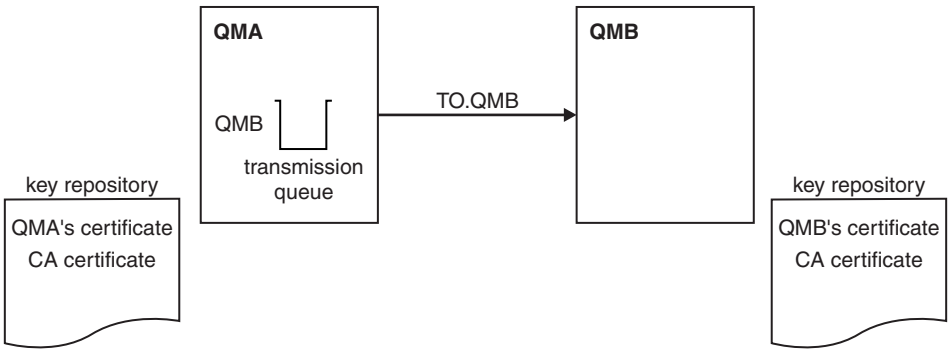


Figure 65. Configuration resulting from this task

In Figure 65, the key repository for QMA contains QMA's certificate and the CA certificate. The key repository for QMB contains QMB's certificate and the CA certificate. In this example both QMA's certificate and QMB's certificate were issued by the same CA. If QMA's certificate and QMB's certificate were issued by different CAs then the key repositories for QMA and QMB must contain both CA certificates.

Procedure

1. Prepare the key repository on each queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Request a CA-signed certificate for each queue manager. You might use different CAs for the two queue managers.
   - On UNIX, Linux, and Windows systems.
3. Add the Certificate Authority certificate to the key repository for each queue manager: If the Queue managers are using different Certificate Authorities then the CA certificate for each Certificate Authority must be added to both key repositories.
   - On UNIX, Linux, and Windows systems.
4. Add the CA-signed certificate to the key repository for each queue manager:
   - On UNIX, Linux, and Windows systems.

5. On QMA, define a sender channel and associated transmission queue by issuing commands like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QMB.MACH.COM) XMITQ(QMB)
SSLCIPH(RC2_MD5_EXPORT) DESCR('Sender channel using SSL from QMA to QMB')

DEFINE QLOCAL(QMB) USAGE(XMITQ)
```

   This example uses CipherSpec RC4_MD5. The CipherSpecs at each end of the channel must be the same.
6. On QMB, define a receiver channel by issuing a command like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(RC2_MD5_EXPORT)
SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL to QMB')
```

   The channel must have the same name as the sender channel you defined in step 6, and use the same CipherSpec.
7. Start the channel:

Results

Key repositories and channels are created as illustrated in Figure 65 on page 390.

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is like that shown in the following examples.

From queue manager QMA, enter the following command:

```
DISPLAY CHS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
     4 : DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QMB)                         CHLTYPE(SDR)
   CONNAME(9.20.25.40)                     CURRENT
   RQMNAME(QMB)
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMB,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(MQGET)
   XMITQ(QMB)
```

From the queue manager QMB, enter the following command:

```
DISPLAY CHS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QMB)                         CHLTYPE(RCVR)
   CONNAME(9.20.35.92)                     CURRENT
   RQMNAME(QMA)
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMA,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
   XMITQ( )
```

In each case, the value of SSLPEER must match that of the Distinguished Name (DN) in the partner certificate that was created in Step 2. The issuer name matches the subject DN of the CA certificate that

signed the personal certificate added in Step 4.

Connecting two queue managers using one-way authentication

Follow these sample instructions to modify a system with mutual authentication to allow a queue manager to connect using one-way authentication to another; that is, when the SSL or TLS client does not send a certificate.

About this task

Scenario:

- Your two queue managers (QM1 and QM2) have been set up as in "Using CA-signed certificates for mutual authentication of two queue managers" on page 368.
- You want to change QM1 so that it connects using one-way authentication to QM2.

The resulting configuration looks like this:

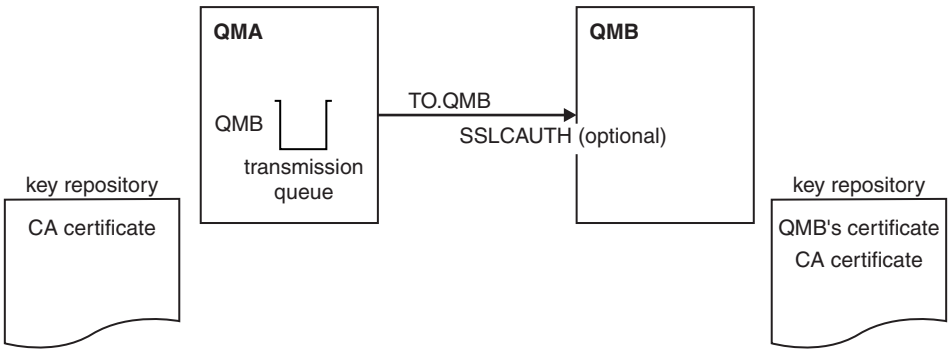


Figure 66. Queue managers allowing one-way authentication

Procedure

1. Remove QM1's personal certificate from its key repository, according to operating system:
   - On UNIX, Linux, and Windows systems. The certificate is labeled as follows:
     - `ibmwebspheremq` followed by the name of your queue manager folded to lower case. For example, for `QM1`, `ibmwebspheremqqm1`.
2. Optional: On QM1, if any SSL or TLS channels have run previously, refresh the SSL or TLS environment.
3. Allow anonymous connections on the receiver.

Results

Key repositories and channels are changed as illustrated in Figure 66

What to do next

If the sender channel was running and you issued the REFRESH SECURITY TYPE(SSL) command (in step 2), the channel restarts automatically. If the sender channel was not running, start it.

At the server end of the channel, the presence of the peer name parameter value on the channel status display indicates that a client certificate has flowed.

Verify that the task has been completed successfully by issuing some DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following examples:

From the QM1 queue manager, enter the following command:

```
DISPLAY CHS(TO.QM2) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
     4 : DISPLAY CHSTATUS(TO.QMB) SSLPEER
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QM2)                         CHLTYPE(SDR)
   CONNAME(9.20.25.40)                     CURRENT
   RQMNAME(QM2)
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMB,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(MQGET)
   XMITQ(QM2)
```

From the QM2 queue manager, enter the following command:

```
DISPLAY CHS(TO.QM2) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(TO.QM2) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QM2)                         CHLTYPE(RCVR)
   CONNAME(9.20.35.92)                     CURRENT
   RQMNAME(QMA)                            SSLCERTI( )
   SSLPEER( )                              STATUS(RUNNING)
   SUBSTATE(RECEIVE)                       XMITQ( )
```

On QM2, the SSLPEER field is empty, showing that QM1 did not send a certificate. On QM1, the value of SSLPEER matches that of the DN in QM2's personal certificate.

Connecting a client to a queue manager securely

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 obtain and manage your digital certificates. On a test system, you can use self-signed certificates or certificates issued by a local certificate authority (CA). On a production system, do not use self-signed certificates. For more information, see ../zs14140_.dita.

For full information about creating and managing certificates, see “Working with SSL or TLS on UNIX, Linux and Windows systems” on page 275.

This collection of topics introduces 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 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 WebSphere 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 WebSphere MQ format, which is `ibmwebspheremq` followed by your logon user ID changed to lowercase, for example `ibmwebspheremqmyuserid`.

WebSphere 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 lowercase.

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 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. For more information about connecting a queue manager anonymously, see "Connecting a client to a queue manager anonymously" on page 375.

Using self-signed certificates for mutual authentication of a client and queue manager

Follow these sample instructions to implement mutual authentication between a client and a queue manager, by using self-signed SSL or TLS certificates.

About this task

Scenario:

- You have a client, C1, and a queue manager, QM1, which need to communicate securely. You require mutual authentication to be carried out between C1 and QM1.
- You have decided to test your secure communication by using self-signed certificates.

DCM on IBM i does not support self-signed certificates, so this task is not applicable on IBM i systems.

The resulting configuration looks like this:

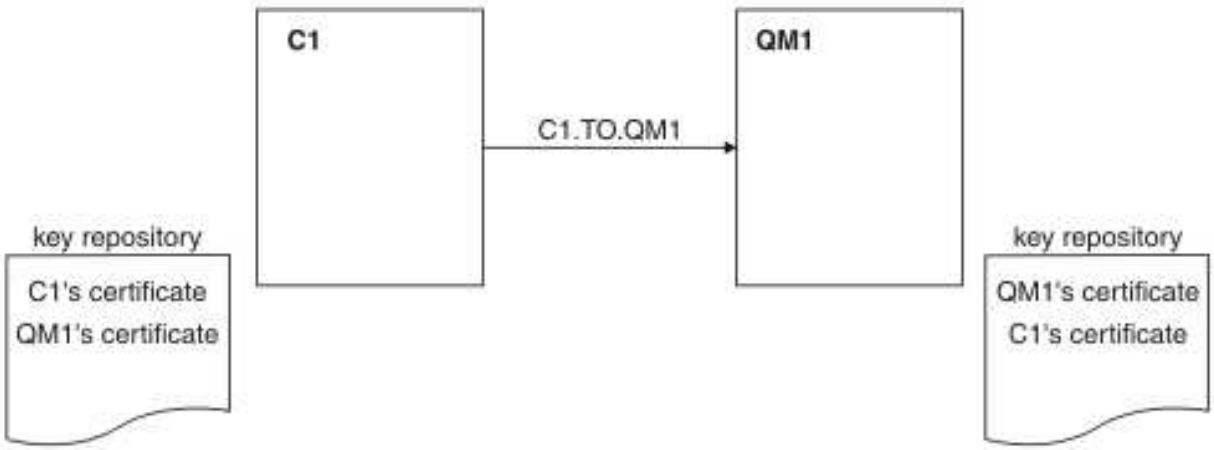


Figure 67. Configuration resulting from this task

In Figure 67, the key repository for QM1 contains the certificate for QM1 and the public certificate from C1. The key repository for C1 contains the certificate for C1 and the public certificate from QM1.

Procedure

1. Prepare the key repository on the client and queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Create self-signed certificates for the client and queue manager:
   - On UNIX, Linux, and Windows systems.
3. Extract a copy of each certificate:

- On UNIX, Linux, and Windows systems.

4. Transfer the public part of the C1 certificate to the QM1 system and vice versa, using a utility such as FTP.
5. Add the partner certificate to the key repository for the client and queue manager:
   - On UNIX, Linux, and Windows systems.
6. Define a client-connection channel in either of the following ways:
   - Using the MQCONNX call with the MQSCO structure on C1, as described in Creating a client-connection channel on the WebSphere MQ MQI client.
   - Using a client channel definition table, as described in Creating server-connection and client-connection definitions on the server.
7. On QM1, define a server-connection channel, by issuing a command like the following example:

   ```
   DEFINE CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCIPH(RC4_MD5_US)
   SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL from C1 to QM1')
   ```

   The channel must have the same name as the client-connection channel you defined in step 6, and use the same CipherSpec.

Results

Key repositories and channels are created as illustrated in Figure 67 on page 394

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following example.

From queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(C1.TO.QM1)                    CHLTYPE(SVRCONN)
   CONNAME(9.20.35.92)                     CURRENT
   SSLCERTI("CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5E:02,CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

It is optional to set the SSLPEER filter attribute of the channel definitions. If the channel definition SSLPEER is set, its value must match the subject DN in the partner certificate that was created in Step 2. After a successful connection, the SSLPEER field in the DISPLAY CHSTATUS output shows the subject DN of the remote client certificate.

Using CA-signed certificates for mutual authentication of a client and queue manager

Follow these sample instructions to implement mutual authentication between a client and a queue manager, by using CA-signed SSL or TLS certificates.

About this task

Scenario:

- You have a client, C1, and a queue manager, QM1, which need to communicate securely. You require mutual authentication to be carried out between C1 and QM1.

- In the future you are planning to use this network in a production environment, and therefore you have decided to use CA-signed certificates from the beginning.

The resulting configuration looks like this:

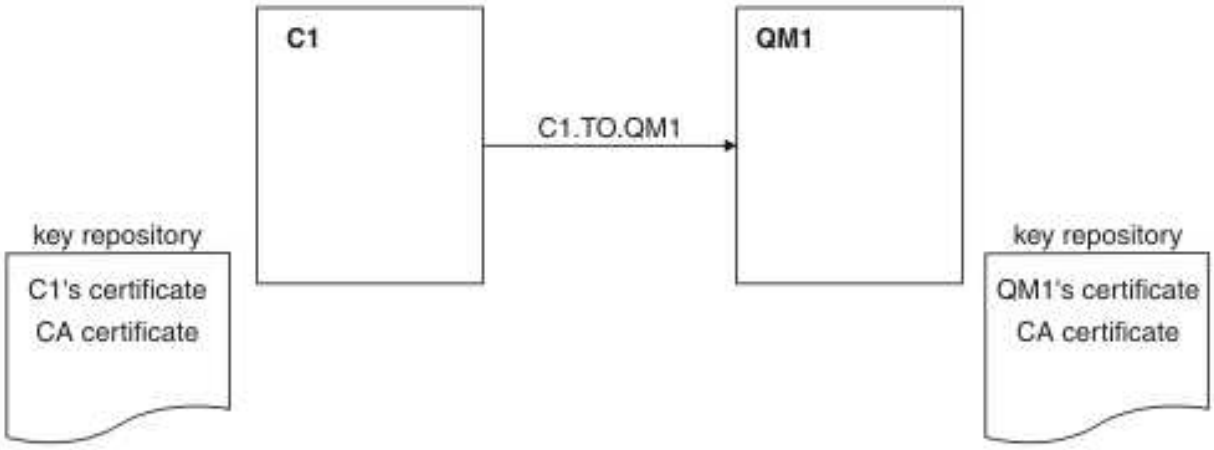


Figure 68. Configuration resulting from this task

In Figure 68, the key repository for C1 contains certificate for C1 and the CA certificate. The key repository for QM1 contains the certificate for QM1 and the CA certificate. In this example both C1's certificate and QM1's certificate were issued by the same CA. If C1's certificate and QM1's certificate were issued by different CAs then the key repositories for C1 and QM1 must contain both CA certificates.

Procedure

1. Prepare the key repository on the client and queue manager, according to operating system:
   - On UNIX, Linux, and Windows systems.
2. Request a CA-signed certificate for the client and queue manager. You might use different CAs for the client and queue manager.
   - On UNIX, Linux, and Windows systems.
3. Add the certificate authority certificate to the key repository for the client and queue manager. If the client and queue manager are using different Certificate Authorities then the CA certificate for each Certificate Authority must be added to both key repositories.
   - On UNIX, Linux, and Windows systems.
4. Add the CA-signed certificate to the key repository for the client and queue manager:
   - On UNIX, Linux, and Windows systems.
5. Define a client-connection channel in either of the following ways:
   - Using the MQCONNX call with the MQSCO structure on C1, as described in Creating a client-connection channel on the WebSphere MQ MQI client.
   - Using a client channel definition table, as described in Creating server-connection and client-connection definitions on the server.
6. On QM1, define a server-connection channel by issuing a command like the following example:

   ```
   DEFINE CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCIPH(RC2_MD5_EXPORT)
   SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL from C1 to QM1')
   ```

   The channel must have the same name as the client-connection channel you defined in step 6, and use the same CipherSpec.

Results

Key repositories and channels are created as illustrated in Figure 68 on page 396.

What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is like that shown in the following example.

From the queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(C1.TO.QM1)                      CHLTYPE(SVRCONN)
   CONNAME(9.20.35.92)                     CURRENT
   SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
   SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMA,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

The SSLPEER field in the DISPLAY CHSTATUS output shows the subject DN of the remote client certificate that was created in Step 2. The issuer name matches the subject DN of the CA certificate that signed the personal certificate added in Step 4.

Connecting a client to a queue manager anonymously

Follow these sample instructions to modify a system with mutual authentication to allow a queue manager to connect anonymously to another.

About this task

Scenario:

- Your queue manager and client (QM1 and C1) have been set up as in "Using CA-signed certificates for mutual authentication of a client and queue manager" on page 373.
- You want to change C1 so that it connects anonymously to QM1.

The resulting configuration looks like this:

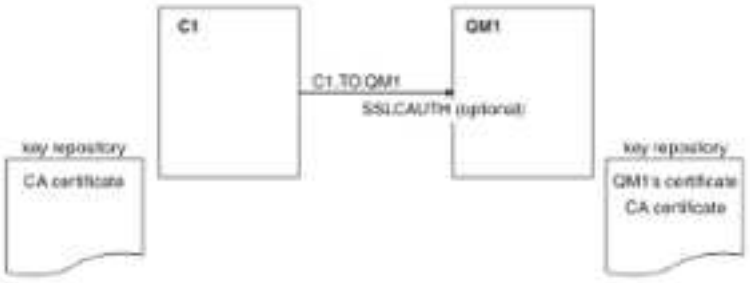


Figure 69. Client and queue manager allowing anonymous connection

Procedure

1. Remove the personal certificate from key repository for C1, according to operating system:
   - On UNIX, Linux, and Windows systems. The certificate is labeled as follows:

– `ibmwebspheremq` followed by your logon user ID folded to lower case, for example `ibmwebspheremqmyuserid`.

2. Restart the client application, or cause the client application to close and reopen all SSL or TLS connections.
3. Allow anonymous connections on the queue manager, by issuing the following command:
   ```
   ALTER CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) SSLCAUTH(OPTIONAL)
   ```

Results

Key repositories and channels are changed as illustrated in Figure 69 on page 397

What to do next

At the server end of the channel, the presence of the peer name parameter value on the channel status display indicates that a client certificate has flowed.

Verify that the task has been completed successfully by issuing some DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following example:

From queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
     5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(C1.TO.QM1)                      CHLTYPE(SVRCONN)
   CONNAME(9.20.35.92)                     CURRENT
   SSLCERTI( )                             SSLPEER( )
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

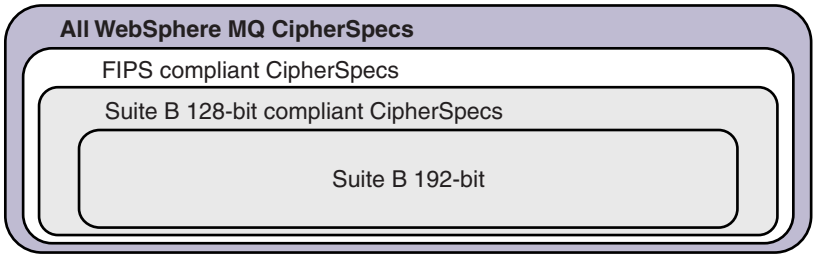
The SSLCERTI and SSLPEER fields are empty, showing that C1 did not send a certificate.

Specifying CipherSpecs

Specify 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 WebSphere MQ are FIPS compliant. Others, such as `NULL_MD5`, are not. Similarly, some of the FIPS compliant CipherSpecs are also Suite B compliant although others, 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:



Cipher specifications that you can use with IBM WebSphere MQ SSL and TLS support 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.

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
NULL_MD5 [a]	SSL 3.0	MD5	None	0	No	No	No
NULL_SHA [a]	SSL 3.0	SHA-1	None	0	No	No	No
RC4_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC4	40	No	No	No
RC4_MD5_US [a]	SSL 3.0	MD5	RC4	128	No	No	No
RC4_SHA_US [a]	SSL 3.0	SHA-1	RC4	128	No	No	No
RC2_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC2	40	No	No	No
DES_SHA_EXPORT [2] [a]	SSL 3.0	SHA-1	DES	56	No	No	No
RC4_56_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	RC4	56	No	No	No
DES_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	DES	56	No	No	No
TLS_RSA_WITH_AES_128_CBC_SHA [a]	TLS 1.0	SHA-1	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA [4] [a]	TLS 1.0	SHA-1	AES	256	Yes	No	No
TLS_RSA_WITH_DES_CBC_SHA [a]	TLS 1.0	SHA-1	DES	56	No[5]	No	No
FIPS_WITH_DES_CBC_SHA [b]	SSL 3.0	SHA-1	DES	56	No[6]	No	No
TLS_RSA_WITH_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	256	Yes	No	No
ECDHE_ECDSA_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No
ECDHE_RSA_RC4_128_SHA256 [b]	TLS 1.2	SHA_1	RC4	128	No	No	No
ECDHE_ECDSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_ECDSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_RSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_RSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_ECDSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	Yes	No
ECDHE_ECDSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	Yes
ECDHE_RSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
ECDHE_RSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
TLS_RSA_WITH_NULL_SHA256 [b]	TLS 1.2	SHA-256	None	0	No	No	No
ECDHE_RSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
ECDHE_ECDSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
TLS_RSA_WITH_NULL_NULL [b]	TLS 1.2	None	None	0	No	No	No
TLS_RSA_WITH_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No

Notes:

1. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
2. 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.
3. The handshake key size is 1024 bits.
4. This CipherSpec cannot be used to secure a connection from the WebSphere MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
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. 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.

Platform support:

- a Available on all supported platforms.
- b Available only on UNIX, Linux, and Windows platforms.

Related concepts:

"Digital certificates and CipherSpec compatibility in IBM WebSphere MQ" on page 192
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 WebSphere MQ.

Related information:

DEFINE CHANNEL

ALTER CHANNEL

Obtaining information about CipherSpecs using WebSphere MQ Explorer

You can use WebSphere MQ Explorer to display descriptions of CipherSpecs.

Use the following procedure to obtain information about the CipherSpecs in "Specifying CipherSpecs" on page 376:

1. Open **WebSphere 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 WebSphere 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 "Specifying CipherSpecs" on page 376. 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* at http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp
- For V5R4, the *IBM i product documentation* at http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp

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 two-character string representing a hexadecimal value. The hexadecimal codes correspond to the values defined in the SSL protocol.

For more information, refer to the description of gsk_environment_open() in the API reference chapter of *z/OS Cryptographic Services System SSL Programming*, SC24-5901, where there is a list of all the supported SSL V3.0 and TLS V1.0 cipher specifications in the form of 2-digit hexadecimal codes.

Considerations for WebSphere MQ clusters

With WebSphere MQ clusters it is safest to use the CipherSpec names in "Specifying CipherSpecs" on page 376. 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 406.

Specifying a CipherSpec for a WebSphere MQ MQI client

You have three options for specifying a CipherSpec for a WebSphere 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 WebSphere MQ classes for Java and WebSphere MQ classes for JMS

WebSphere MQ classes for Java and WebSphere MQ classes for JMS specify CipherSuites differently from other platforms.

For information about specifying a CipherSuite with WebSphere MQ classes for Java, see Secure Sockets Layer (SSL) support

For information about specifying a CipherSuite with WebSphere MQ classes for JMS, see Using Secure Sockets Layer (SSL) with WebSphere MQ classes for JMS

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 IBM WebSphere 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 WebSphere 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 WebSphere 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 your platform, authorize access to connect to the queue manager and to put to the queue on that queue manager.

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 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
```

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.

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 351.

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.

   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.

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, see WebSphere MQ support for SSL and TLS. The advice there is generally applicable to cluster channels, but you might want to give some special consideration to the following:

In a IBM WebSphere 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 WebSphere 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=WebSphere 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 WebSphere 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 WebSphere 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 one CLUSRCVR at a time, and allow the changes to flow through the cluster before changing the next. Make sure that you do not change the reverse path until the changes for the current channel have been distributed throughout the cluster.

Procedure

1. Switch the CLUSRCVR channels to SSL in any order you like. The changes flow in the opposite direction over channels which are not changed to SSL.
2. 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.

Related concepts:

"Specifying CipherSpecs" on page 376
Specify 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 WebSphere MQ" on page 192
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 WebSphere MQ.

Related information:

Clustering: Using REFRESH CLUSTER best practices

Disabling SSL or 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 . You can turn off SSL or 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 SSL or 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 SSL or 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 70 on page 409 and described in the following example.

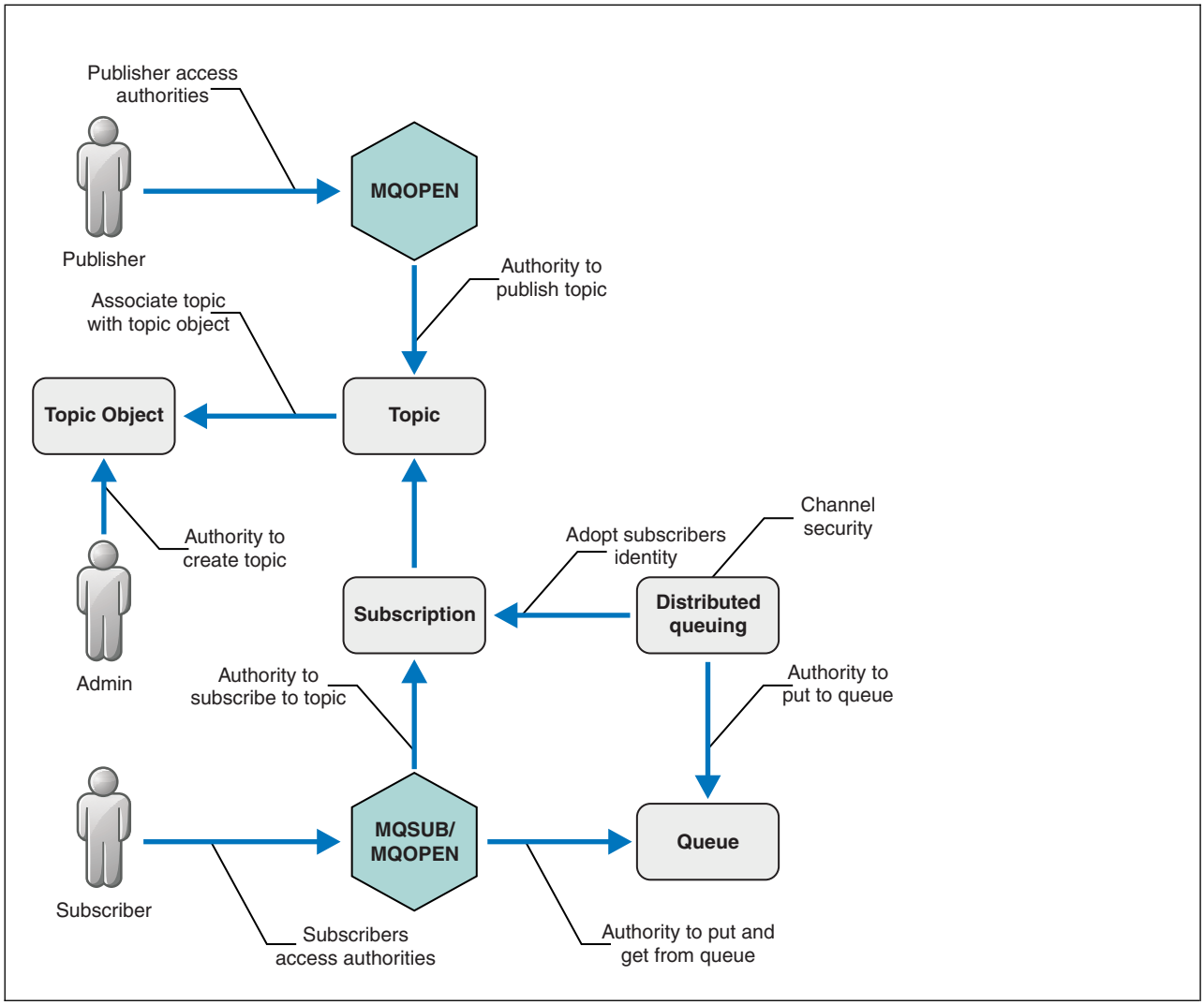


Figure 70. Publish/subscribe security relationships

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 411 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 416 and "Grant access to a user to publish to a topic" on page 421.

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 412

Subscribe using a topic

"Subscribing using a topic string where the topic node does not exist" on page 413

Subscribe using a topic with wildcards

"Subscribing using a topic string that contains wildcard characters" on page 413

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 425.

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 414.

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

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 WebSphere 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 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. WebSphere 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:

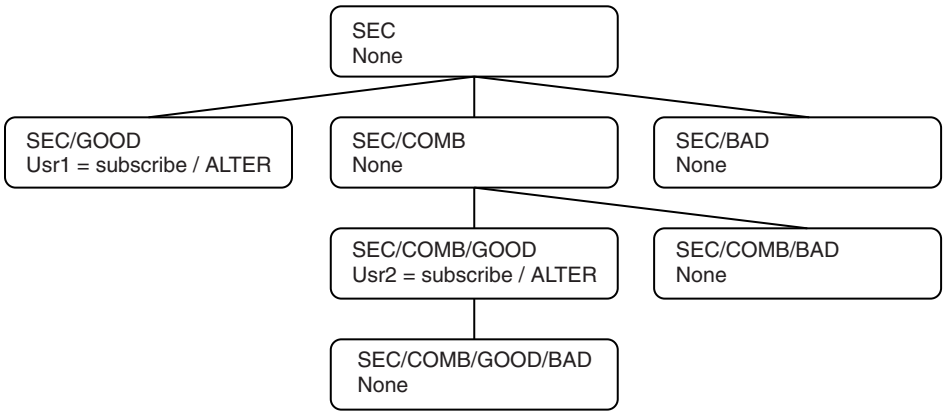
Table 21. Example topic object authorities

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`

Table 21. Example topic object authorities (continued)

Topic name	Topic string	Authorities - not z/OS	z/OS authorities
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.

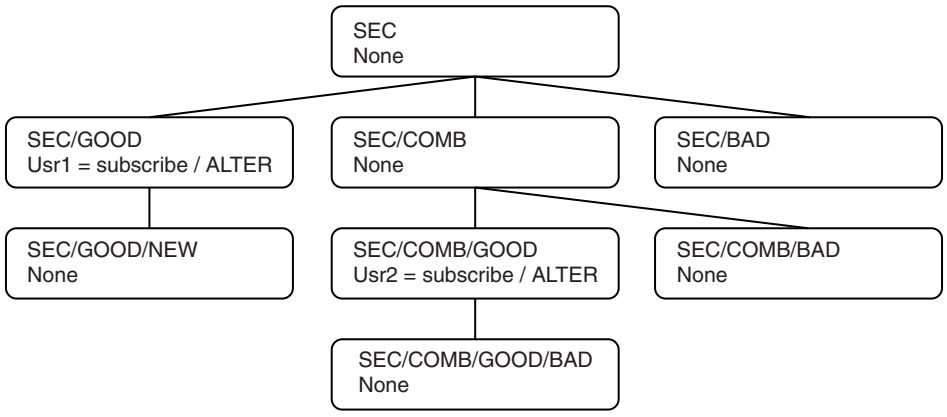
Subscribing using a topic string where the topic node exists

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 22:

Table 22. User IDs used for security checks for commands

Command	SUBUSER specified and blank	SUBUSER specified and completed	SUBUSER not specified
DEFINE	Use the administrator ID	Use the user ID specified in **SUBUSER**	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.

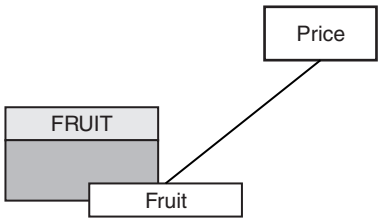


Figure 71. Topic object access example

Table 23. 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:
   - 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
     ```

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:

- 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"
```

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 416.

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.

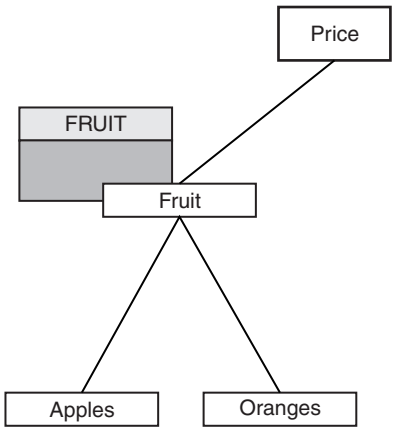


Figure 72. Example of granting access to a topic within a topic tree

Table 24. 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 417.

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.

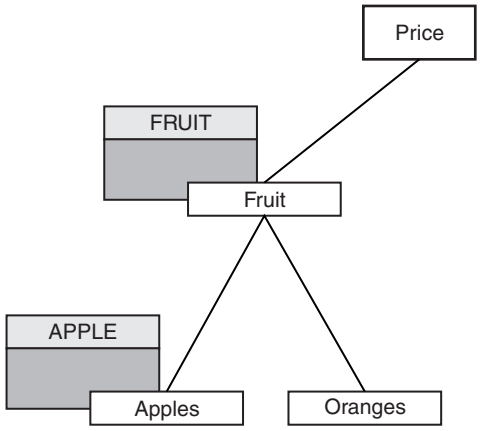


Figure 73. Granting access to specific topics within a topic tree

Table 25. 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 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:
   - 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
     ```

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:

- 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"
  ```

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 418 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.

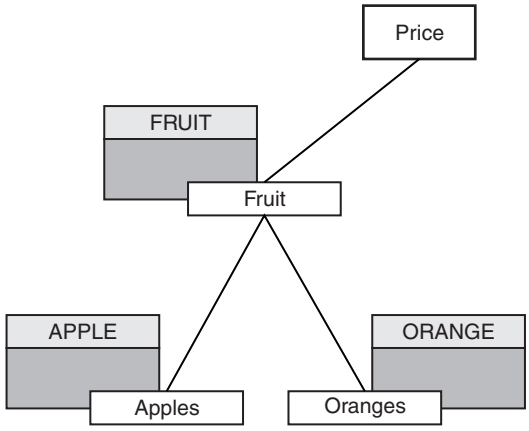


Figure 74. Example of granting access control to avoid additional messages.

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:
   - 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
     ```

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:

- 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"
```

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:

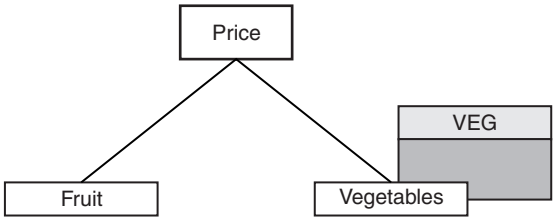


Figure 75. Granting publish access to a topic

Table 26. 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:
   - 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
     ```

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:

- 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"
```

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 421.

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.

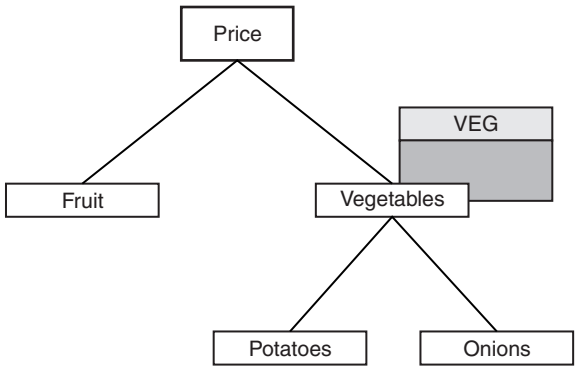


Figure 76. Granting publish access to a topic within a topic tree

Table 27. 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 422.

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.

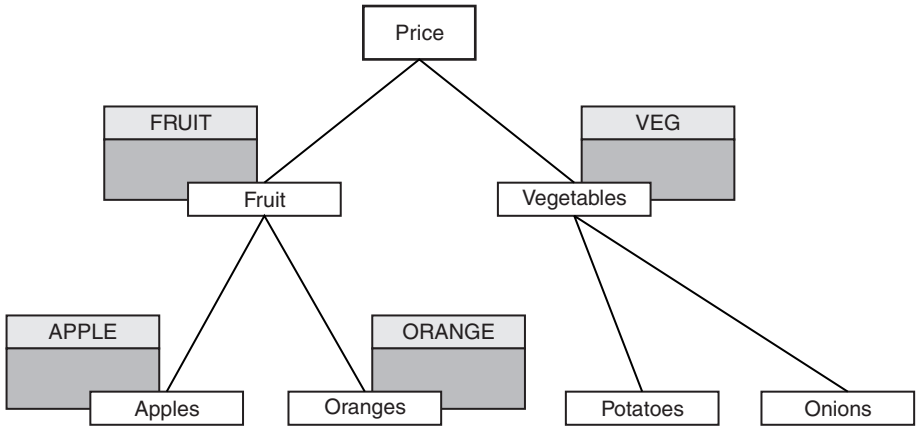


Figure 77. Granting access for publishing and subscribing

Table 28. 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:

- 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
```

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:

- 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"
```

Following the complete set of these tasks, gives `USER1` and `USER2` the following access authorities for publish and subscribe to the topics listed:

Table 29. 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 30. 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.

WebSphere MQ Advanced Message Security

IBM WebSphere MQ Advanced Message Security (WebSphere MQ AMS) is a separately licensed component of WebSphere MQ that provides a high level of protection for sensitive data flowing through the WebSphere MQ network, while not impacting the end applications.

WebSphere MQ AMS overview

WebSphere MQ applications can use WebSphere 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 between version 7.0.1 and version 7.5

As IBMWebSphere MQ Advanced Message Security became a component in WebSphere MQ 7.5, some aspects of WebSphere MQ AMS functionality have changed, what might affect existing applications, administrative scripts, or management procedures.

Review the following list of changes carefully before upgrading queue managers to version 7.5. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems to WebSphere MQ version 7.5:

- WebSphere MQ AMS installation is a part of WebSphere MQ installation process.
- WebSphere MQ AMS security capabilities are enabled with its installation and controlled with security policies. You do not need to enable interceptors to allow WebSphere MQ AMS start intercepting data.
- WebSphere MQ AMS in WebSphere MQ version 7.5 does not require the use of the **cfgmqs** command as in the stand-alone version of WebSphere MQ AMS.

Features and functions of WebSphere MQ Advanced Message Security V7.5

WebSphere MQ Advanced Message Security expands WebSphere 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, WebSphere MQ AMS verifies that a sender of message data is authorized to place signed messages on a target queue.

Here is a complete list of WebSphere MQ AMS functions:

- Secures sensitive or high-value transactions processed by WebSphere 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 WebSphere MQ.

Error handling

WebSphere 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 WebSphere 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.

WebSphere MQ AMS uses the SYSTEM.PROTECTION.ERROR.QUEUE as its error handling queue. All messages put by WebSphere MQ AMS to the SYSTEM.PROTECTION.ERROR.QUEUE are preceded by MQDLH header.

Your WebSphere 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 WebSphere 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 **distinguished name (DN)** field in a certificate associated with signed or encrypted messages. WebSphere 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 450
To provide transparent cryptographic protection to WebSphere MQ applications, WebSphere MQ Advanced Message Security uses the keystore file, where public key certificates and a private key are stored.

Digital certificates:

WebSphere 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: WebSphere 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:

The Object Authority Manager (OAM) is the authorization service component supplied with the WebSphere MQ products.

The access to WebSphere MQ Advanced Message Security entities is controlled through WebSphere 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 WebSphere 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

WebSphere MQ Advanced Message Security depends on several technology components to provide a security infrastructure.

WebSphere MQ Advanced Message Security supports the following WebSphere MQ application programming interfaces (APIs):

- Message queue interface (MQI)
- WebSphere MQ Java Message Service (JMS) 1.0.2 and 1.1.
- WebSphere MQ Base Classes for Java
- WebSphere MQ classes for .Net in an unmanaged mode

Note: WebSphere MQ Advanced Message Security supports X.509 compliant certificate authorities.

Known limitations:

Learn about limitations of IBM WebSphere MQ Advanced Message Security.

- The following IBM WebSphere 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 WebSphere 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.
- All Java applications are dependent on the IBM Java Runtime.

  IBM WebSphere MQ Advanced Message Security does not support JRE provided by other vendors.
- JMS and Java client applications using IBM WebSphere MQ Advanced Message Security in client mode.

  Any JMS, or Java, client application (including IBM WebSphere MQ Explorer and IBM WebSphere MQ Managed File Transfer agents) cannot use IBM WebSphere MQ Advanced Message Security in client mode with a WebSphere MQ queue manager earlier than Version 7.5.

  In order to use message protection policies, these applications either need to interact with an IBM WebSphere MQ Version 7.5 queue manager, or connect in local bindings mode to a queue manager on the same machine as the application.
- You should avoid putting two or more certificates with the same Distinguished Names, in a single keystore file, because the IBM WebSphere MQ Advanced Message Security intereceptor's functioning with such certificates is undefined.
- The IBM WebSphere MQ Version 7.5 resource adapter does not support IBM WebSphere MQ Advanced Message Security. If message protection is required to be used with IBM WebSphere MQ classes for JMS or IBM WebSphere MQ classes for Java applications running within an application server environment then:
  - Either the application server must be configured to use the Version 8.0 or later resource adapter.
  - Or Message Channel Agent (MCA) interception must be used.

User scenarios

Familiarize yourself with possible scenarios to understand what business goals you can achieve with WebSphere MQ Advanced Message Security.

Quick Start Guide for Windows platforms:

Use this guide to quickly configure IBM WebSphere 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 WebSphere 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. WebSphere MQ Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the WebSphere MQ infrastructure through the standard WebSphere MQ interface. The basic setup is done in WebSphere MQ and is configured in the following steps.

You can use WebSphere 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 \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 434.

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 Java clients" on page 440 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 WebSphere 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 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 " 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 451.
- *%HOMEDRIVE%\%HOMEPATH%*\.mqs\keystore.conf (eg. C:\Documents and Settings\alice\.mqs\keystore.conf) is the default location where WebSphere 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 450.
- 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
   ```

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_pub
   ```

3. Repeat steps for bob:

   ```
   runmqakm -cert -extract -db "C:/Documents and Settings/alice/AMS/bobkey.kdb" -pw passw0rd -label Bob_Cert -target bob

   runmqakm -cert -add -db "C:/Documents and Settings/bob/AMS/alicekey.kdb" -pw passw0rd -label Bob_Cert -file bob_publi
   ```

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 UNIX platforms:

Use this guide to quickly configure IBM WebSphere 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 WebSphere MQ components for Linux systems
- IBM WebSphere MQ components for HP-UX systems
- IBM WebSphere MQ components for AIX systems
- IBM WebSphere 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. WebSphere MQ Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the WebSphere MQ infrastructure through the standard WebSphere MQ interface. The basic setup is done in WebSphere MQ and is configured in the following steps.

You can use WebSphere 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 440.

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 Java clients" on page 440 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 WebSphere 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 451.
- `HOME/.mqs/keystore.conf` is the default location where WebSphere 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 450.

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 Java clients:

Use this guide to quickly configure IBM WebSphere 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 key store 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. WebSphere MQ Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the WebSphere MQ infrastructure through the standard WebSphere MQ interface. The basic setup is done in WebSphere 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** 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 our 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 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 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 444.

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 to create your keystore in, 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 will be 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 certifcates 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 WebSphere 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` will be 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` will be 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` because you have followed **Quick Start Guide** (Windows or UNIX), you can edit the existing one to add in the above 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 WebSphere 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 WebSphere 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 WebSphere 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, WebSphere 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 WebSphere MQ to handle it. After WebSphere MQ Advanced Message Security has processed the message put into a remote queue, WebSphere 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, WebSphere 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 WebSphere MQ Advanced Message Security roll-out has been completed.

Routing protected messages using WebSphere Message Broker:

IBM WebSphere MQ Advanced Message Security can protect messages in an infrastructure where 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 WebSphere Message Broker environment.

About this task

WebSphere 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 WebSphere Message Broker, you can either allow WebSphere Message Broker 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 - Message Broker cannot see message content:
Before you begin

You should have your WebSphere Message Broker 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), [1] (QCECIL), or the default (QDEF) queue. The routing is possible because WebSphere MQ Advanced Message Security protects only the message payload and not its headers and properties which remain unprotected and can be read by WebSphere Message Broker. WebSphere MQ Advanced Message Security is used only by *alice*, *bob* and *cecil*. It is not necessary to install or configure it for the WebSphere Message Broker.

WebSphere Message Broker 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 WebSphere Message Broker 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). WebSphere Message Broker puts the routed message to an unprotected alias. The recipients retrieve the message from a protected output queue where WebSphere MQ AMS will transparently decrypt the message.

Procedure

1. Configure *alice*, *bob* and *cecil* to use WebSphere 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)
   ```
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 WebSphere Message Broker 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.

1. cecil's

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 WebSphere Message Broker 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 WebSphere Message Broker from the AIN alias queue. WebSphere Message Broker decides where to route the message reading the routeTo property which is, as all properties, not encrypted. WebSphere Message Broker 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 - Message Broker can see message content:
About this task

In this scenario, a group of individuals are allowed to send messages to WebSphere Message Broker. Another group are authorized to receive messages which are created by WebSphere Message Broker. The transmission between the parties and WebSphere Message Broker cannot be eavesdropped.

Remember that WebSphere Message Broker 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 WebSphere Message Broker is considered an authorized party allowed to read or sign the message payload, you must configure WebSphere MQ Advanced Message Security for the user starting the WebSphere Message Broker service. Be aware it is not necessarily the same user who puts/gets the messages onto queues nor the user creating and deploying the WebSphere Message Broker applications.

Procedure

1. Configure *alice*, *bob*, *cecil* and *dave* and the WebSphere Message Broker service user, to use WebSphere 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 WebSphere Message Broker service user.

   Do this by extracting to external files each of the certificates identifying *alice*, *bob*, *cecil* and *dave*, then adding the extracted certificates to the WebSphere Message Broker 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 WebSphere Message Broker 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 WebSphere Message Broker service user's certificate to encrypt the messages correctly. The WebSphere Message Broker service user needs *alice's* and *bob's* certificates to verify authors of the messages. The WebSphere Message Broker service user needs *cecil's* and *dave's* certificates to encrypt the messages for them. *cecil* and *dave* need the WebSphere Message Broker service user's certificate to verify if the message comes from WebSphere Message Broker.
4. Define a local queue named `IN` and define the security policy with *alice* and *bob* specified as authors and WebSphere Message Broker's service user 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 WebSphere Message Broker's service user 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 WebSphere Message Broker 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 WebSphere Message Broker 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 WebSphere Message Broker to read it. WebSphere Message Broker will only accept messages from *alice* and *bob* and will reject any others. The accepted messages will be 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 WebSphere Message Broker are rejected.

Using WebSphere MQ AMS with WebSphere MQ Managed File Transfer:

This scenario explains how to configure WebSphere MQ Advanced Message Security to provide message privacy for data being sent through a WebSphere MQ Managed File Transfer.

Before you begin

Ensure that you have WebSphere MQ Advanced Message Security component installed on the WebSphere MQ installation hosting the queues used by WebSphere MQ Managed File Transfer that you wish to protect.

If your WebSphere 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 WebSphere MQ Managed File Transfer agents is interrupted, possibly confidential data might remain unprotected on the underlying WebSphere MQ queues used to manage the transfer. This scenario explains how to configure and use WebSphere MQ Advanced Message Security to protect such data on the WebSphere MQ Managed File Transfer queues.

In this scenario we consider a simple topology comprising one machine with two WebSphere MQ Managed File Transfer queues. agents, `AGENT1` and `AGENT2` sharing a single queue manager, `hubQM`, as is described in the scenario Basic file transfer using the scripts. 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 WebSphere MQ Managed File Transfer agent processes. If you are using your own user and group names, change the commands accordingly.

About this task

WebSphere MQ Advanced Message Security uses public key cryptography to sign and/or encrypt messages on protected queues.

Note:

- If your WebSphere 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 WebSphere 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 Java clients" on page 440.

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Advanced Message Security

Install IBM WebSphere MQ Advanced Message Security component on various platforms.

About this task

For complete installation procedures, see Installing WebSphere MQ Advanced Message Security.

Related tasks:

Uninstalling WebSphere MQ Advanced Message Security

Using keystores and certificates

To provide transparent cryptographic protection to WebSphere MQ applications, WebSphere MQ Advanced Message Security uses the keystore file, where public key certificates and a private key are stored.

In WebSphere 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.

Location of the keystore is provided in the keystore configuration file, which is keystore.conf by default. Each WebSphere MQ Advanced Message Security user must have the keystore configuration file that points to a keystore file. WebSphere 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: $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 462
The sender distinguished names (DNs) identify users who are authorized to place messages on a queue.

"Recipient distinguished names" on page 463
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 WebSphere MQ Advanced Message Security to the location of the appropriate keystore.

There are two types of configuration CMS and Java (JKS and JCEKS). CMS configuration entries are prefixed with `cms.` and Java are prefixed with `jks.` or `jceks.` depending on the type of a keystore.

The configuration file, depending on the type of the configuration file, can have one of the following structures:

```
cms.keystore = /<dir>/<keystore_file>
cms.certificate = certificate_label

jceks.keystore = <dir>/Keystore
jceks.certificate = <certificate_label>
jceks.encrypted = no
jceks.keystore_pass = <password>
jceks.key_pass = <password>
jceks.provider = IBMJCE

jks.keystore = <dir>/Keystore
jks.certificate = <certificate_label>
jks.encrypted = no
jks.keystore_pass = <password>
jks.key_pass = <password>
jks.provider = IBMJCE
```

Configuration file parameters are defined as follows:

keystore
: Path to the keystore file.

  Important:

  - The path to the keystore file must not include the file extension.
  - For Java keystore files, WebSphere MQ AMS supports the following file formats: `.jks`, `.jceks`, `.jck`.

certificate
: Certificate label.

encrypted
: Status of the password.

keystore_pass
: Password for the keystore file.

  Note:

  - For the CMS keystore, WebSphere 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.
  - Storing passwords in plain text is a security risk.

key_pass
: Password for the user's private key.

Important: Storing passwords in plain text form can pose a security risk.

provider
The Java security provider that implements cryptographic algorithms required by the keystore certificate.

Note: Currently IBMJCE is the only provider that is supported by WebSphere MQ Advanced Message Security.

Important: Information stored in the keystore is crucial for the secure flow of data sent using WebSphere MQ, which is why security administrators must pay particular attention when assigning file permissions to these files.

For more information about managing the keystore using GSKit commands, see GSKCapiCmd program users guide.

Here is an 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 = <password>
jceks.key_pass = <password>
jceks.provider = IBMJCE
```

Related tasks:

"Protecting passwords in Java" on page 460
Storing keystore and private key passwords as plain text poses a security risk so WebSphere 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 WebSphere MQ to selectively enable policies to be applied for server connection channels.

MCA interception allows clients that remain outside WebSphere MQ AMS to still be connected to a queue manager and their messages to be encrypted and decrypted.

MCA interception is intended to provide WebSphere MQ AMS capability when WebSphere MQ AMS cannot be enabled at the client. Note that using MCA interception and an WebSphere MQ AMS-enabled client leads to double-protection of messages which might be problematic for receiving applications. For more information, see "Environment variable used to disable WebSphere MQ AMS at the client" on page 454.

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 WebSphere MQ AMS keystore, see "Using keystores and certificates" on page 450.

To enable MCA interception, you must provide the name of a channel that you want to use in the keystore configuration file. For MCA interception, only a cms keystore type can be used.

For an example of setting up MCA interception, see "WebSphere MQ AMS MCA interception example."

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.

WebSphere MQ AMS MCA interception example:

An example task on how you setup an WebSphere 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.

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, WebSphere 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/mq75
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 WebSphere MQ Version 7.5 installation:

```
/opt/mq75/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 WebSphere MQ Version 7.5 installation:

```
/opt/mqm/samp/bin/amqsgetc TESTQ TESTQMGR
```

Related tasks:

"Quick Start Guide for Java clients" on page 440
Use this guide to quickly configure IBM WebSphere 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 key store to verify user identities, and defined signing/encryption policies for your queue manager.

Related reference:

"Known limitations" on page 430
Learn about limitations of IBM WebSphere MQ Advanced Message Security.

Environment variable used to disable WebSphere MQ AMS at the client

Disable IBM WebSphere 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 WebSphere MQ Advanced Message Security by setting an environment variable. For C clients, you can also disable IBM WebSphere MQ Advanced Message Security by using the `DisableClientAMS` property under the **`Security`** stanza.

You are using Version 7.5 client, or later, and you are attempting to connect to a queue manager from an earlier version of the product, for example, 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 Version 7.5, or later, IBM WebSphere MQ classes for Java or IBM WebSphere MQ classes for JMS client.

To set the AMQ_DISABLE_CLIENT_AMS environment variable, you must be running 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 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 WebSphere MQ Advanced Message Security protected queues, see "Problems opening protected queues when using JMS" on page 478.

DisableClientAMS property in the mqclient.ini file

For C clients from Version 7.5.0, Fix Pack 5 or later, you can also disable or enable IBM WebSphere MQ Advanced Message Security at the client, in the `mqclient.ini` file, by using the property name `DisableClientAMS`, under the **`Security`** stanza.

- To disable IBM WebSphere MQ Advanced Message Security:

```
Security:
DisableClientAMS=Yes
```

- To enable IBM WebSphere MQ Advanced Message Security:

```
Security:
DisableClientAMS=No
```

Key usage extensions

Key usage extensions place additional restrictions on the way a certificate can be used.

In WebSphere 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 465
WebSphere MQ Advanced Message Security data-protection policies imply a quality of protection (QOP).

Certificate validation methods in WebSphere MQ AMS

You can use WebSphere 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.

WebSphere MQ AMS allows you to verify a certificate validity by using either Online Certificate Status Protocol (OCSP) or certificate revocation list (CRL).

WebSphere MQ AMS can be configured for either OCSP or CRL checking or both. If both methods are enabled, then, for performance reasons, WebSphere MQ AMS uses OCSP for revocation status first. If the revocation status of a certificate is undetermined after the OCSP checking, WebSphere MQ AMS uses the CRL checking.

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.

"Certificate revocation lists (CRLs)" on page 458
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.

Enabling OCSP checking in native interceptors:

To enable Online Certificate Status Protocol (OCSP) checking in WebSphere MQ Advanced Message Security, you must modify the keystore configuration file.

Procedure

Add the following options to the keystore configuration file:

Note: Values for individual options provided in the table are default.

You must specify one of the following values:

- `ocsp.enable=on`
- `ocsp.url=<reposnder_URL>`
- `ocsp.http.proxy.host=<OCSP_proxy>`

Option	Description
`ocsp.enable=off`	Enable the OCSP checking if the certificate being checked has a Authority Info Access Extension with an PKIX_AD_OCSP access method containing a URI of where the OCSP Responder is located. Possible values: `on/off`.
`ocsp.url=<reposnder_URL>`	The URL address of OCSP responder.
`ocsp.http.proxy.host=<OCSP_proxy>`	The URL address of the OCSP proxy server.
`ocsp.http.proxy.port=<port_number>`	The OCSP proxy server's port number.
`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.

Option	Description
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 WebSphere MQ Advanced Message Security times-out.

Enabling OCSP checking in Java:

To enable OCSP checkin for Java in WebSphere 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 WebSphere MQ Advanced Message Security:

Using java.security:

Check whether your certificate has Authority Information Access (AIA) set up.

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, WebSphere 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.WebSphere MQ AMS verifies the certificate path using a PKIX validation algorithm. When the chain is created and verified, WebSphere 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, WebSphere MQ AMS verifies whether **digitalSignature** or **nonRepudiation** are also set. If they are not, the `MQRC_SECURITY_ERROR` is reported and logged. Next, WebSphere 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 WebSphere MQ AMS. If no CRL related configuration is found in the keystore configuration file, WebSphere MQ AMS performs no CRL validity check. For each CA certificate, WebSphere 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 have to modify the keystore configuration file so that WebSphere MQ Advanced Message Security can download CLRs from the Lightweight Directory Access Protocol (LDAP) server.

Procedure

Add the following options to the configuration file:

Note: Values for individual options provided in the table are default.

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 WebSphere 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.

Option	Description
`crl.ldap.version=3`	LDAP protocol version number. Possible values: 2 or 3.
`crl.ldap.user=cn=<username>`	Login 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 a HTTP server that is accepted by GSKit.
`crl.http.timeout=30`	Waiting time for a server response, in seconds, after which WebSphere MQ AMS time-outs.
`crl.http.cache_size=0`	HTTP cache size, in bytes.

Enabling certificate revocation list support in Java:

To enable CRL support in WebSphere MQ Advanced Message Security, you must modify the keystore configuration file to allow WebSphere 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 WebSphere 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.

Protecting passwords in Java

Storing keystore and private key passwords as plain text poses a security risk so WebSphere 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 450.

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
```

Security policies

WebSphere 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 WebSphere 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 465
WebSphere MQ Advanced Message Security data-protection policies imply a quality of protection (QOP).

"Security policy attributes" on page 464
You can use WebSphere 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 WebSphere 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 a WebSphere MQ Advanced Message Security (WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Advanced Message Security (WebSphere 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, WebSphere 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.
- WebSphere 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 WebSphere MQ AMS plug-in to WebSphere MQ.

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.

Component name	Value
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.
- WebSphere 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 WebSphere MQ Advanced Message Security plug-in to WebSphere MQ.

Security policy attributes:

You can use WebSphere 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 WebSphere 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 WebSphere 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`

WebSphere 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:

WebSphere MQ Advanced Message Security data-protection policies imply a quality of protection (QOP).

The three quality of protection levels in WebSphere 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. WebSphere 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.

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 467
You can use WebSphere MQ Advanced Message Security to alter details of security policies that you have already defined.

"Displaying and dumping security policies" on page 467
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 469
To remove security policies in WebSphere MQ Advanced Message Security, you must use the `setmqspl` command.

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 WebSphere MQ objects.
- You must have the necessary `+connect +inq +chg` authorities to create a security policy. For the complete syntax of authorization change command, see **setmqaut**.
- Make sure that you have necessary permissions to operate on WebSphere MQ queues and queue managers. For more information, see "Granting OAM permissions" on page 471

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 WebSphere 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 necessary `+connect +inq +chg` authorities to create security policies. For the complete syntax of authorization change command, see **setmqaut**.

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 necessary `+connect +inq +dsp` permissions applied to a queue manager to display and dump security policies. For the complete syntax of authorization change command, see **setmqaut**.

About this task

Here is the list of `dspmqspl` command flags:

Table 31. 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 > 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` WebSphere MQ administration group.
  2. Issue `. policies.sh`.

Related reference:
Complete list of dspmqspl command attributes

Removing security policies:

To remove security policies in WebSphere 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 necessary `+connect +inq +chg` authorities to create security policies. For the complete syntax of authorization change command, see **setmqaut**.

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 WebSphere MQ and its ancillary applications. Whenever a queue manager is created, a system queue is also created to store WebSphere MQ internal messages and data. You can protect system queues with WebSphere 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 466.

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
```

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 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 authorised signers, command messages that do not pass signer Distinguished Name (DN) check are not executed by the command server and are not routed to the WebSphere MQ Advanced Message Security error handling queue. Messages that are sent as replies to WebSphere MQ Explorer temporary dynamic queues are not protected by WebSphere MQ AMS.

Changes to WebSphere MQ Advanced Message Security security policies require you to restart the WebSphere MQ command server

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
- 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
- SYSTEM.BROKER.CONTROL.QUEUE
- SYSTEM.BROKER.DEFAULT.STREAM
- SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS
- SYSTEM.CHANNEL.INITQ
- SYSTEM.CHANNEL.SYNCQ
- SYSTEM.CICS.INITIATION.QUEUE
- SYSTEM.CLUSTER.COMMAND.QUEUE
- SYSTEM.CLUSTER.HISTORY.QUEUE
- SYSTEM.CLUSTER.REPOSITORY.QUEUE
- SYSTEM.CLUSTER.TRANSMIT.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
- SYSTEM.PENDING.DATA.QUEUE
- SYSTEM.PROTECTION.ERROR.QUEUE
- SYSTEM.PROTECTION.POLICY.QUEUE
- 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 WebSphere 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 WebSphere 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
```

Command and configuration events

With WebSphere 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 WebSphere MQ are message of the PCF format sent to dedicated queues.

Configuration events messages are sent to the SYSTEM.ADMIN.CONFIG.EVENT queue on the queue manager where the event occurs.

Command events messages are sent to the SYSTEM.ADMIN.COMMAND.EVENT queue on the queue manager where the event occurs.

Events are generated regardless of tools you are using to manage WebSphere MQ Advanced Message Security security policies.

In WebSphere MQ Advanced Message Security, there are four types of events generated by different actions on security policies:

- "Creating security policies" on page 466, which generate two WebSphere MQ event messages:
  - A configuration event
  - A command event
- "Changing security policies" on page 467, which generates three WebSphere 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 467, which generates one WebSphere MQ event message:
  - A command event
- "Removing security policies" on page 469, which generates two WebSphere 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
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.

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 WebSphere 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 WebSphere MQ 7.5 `cmqcfc.h`
Returned: Always.

CommandData contains PCF elements that comprised the PCF command.

Configuration event message format:

Configuration events are PCF messages of standard WebSphere MQ Advanced Message Security format.

Possible values for the MQMD message descriptor can be found in the Event message MQMD (message descriptor) section of the WebSphere MQ product documentation.

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 the Event message MQCFH (PCF header) section of the WebSphere MQ product documentation.

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 WebSphere 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
WebSphere MQ Advanced Message Security protection policy. **1019** - a numeric value defined in WebSphere MQ 7.5 or in the `cmqc.h` file.
Returned: Always.

PolicyName

Description: The WebSphere MQ Advanced Message Security policy name.
Identifier: **MQCA_POLICY_NAME**.
Data type: MQCFST.
Value: **2112** - a numeric value defined in WebSphere MQ 7.5 or in the `cmqc.h` file.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always.

PolicyVersion

Description: The WebSphere MQ Advanced Message Security policy version.
Identifier: **MQIA_POLICY_VERSION**
Data type: MQCFIN
Value **238** - a numeric value defined in WebSphere MQ 7.5 or in the `cmqc.h` file.
Returned: Always

TolerateFlag

Description: The WebSphere MQ Advanced Message Security policy toleration flag.
Identifier: **MQIA_TOLERATE_UNPROTECTED**
Data type: MQCFIN
Value **235** - a numeric value defined in WebSphere MQ 7.5 or in the `cmqc.h` file.
Returned: Always.

SignatureAlgorithm

Description: The WebSphere MQ Advanced Message Security policy signature algorithm.
Identifier: **MQIA_SIGNATURE_ALGORITHM**
Data type: MQCFIN
Value: **236** - a numeric value defined in WebSphere MQ 7.5 or in the `cmqc.h` file.
Returned: Whenever there is a signature algorithm defined in the WebSphere MQ Advanced Message Security policy

EncryptionAlgorithm

Description: The WebSphere MQ Advanced Message Security policy encryption algorithm.
Identifier: **MQIA_ENCRYPTION_ALGORITHM**
Data type: MQCFIN
Value: **237** - a numeric value defined in WebSphere MQ 7.5 or in the `cmqc.h` file.
Returned: Whenever there is an encryption algorithm defined in the WebSphere MQ policy

SignerDNs

Description: Subject DistinguishedName of allowed signers.
Identifier: **MQCA_SIGNER_DN**
Data type: MQCFSL
Value: **2113** - a numeric value defined in WebSphere MQ 7.5 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 WebSphere MQ policy.

RecipientDNs

Description: Subject DistinguishedName of allowed signers.
Identifier: **MQCA_RECIPIENT_DN**
Data type: MQCFSL
Value: **2114** - a numeric value defined in WebSphere MQ 7.5 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 WebSphere MQ policy.

Problems and solutions

This section describes how to solve problems that might arise with any installation of IBM Use this information to identify and resolve any problems relating to WebSphere MQ Advanced Message Security.

com.ibm.security.pkcsutil.PKCSException: Error encrypting contents

Error `com.ibm.security.pkcsutil.PKCSException: Error encrypting contents` suggests that IBM WebSphere MQ Advanced Message Security has problems with accessing cryptographic algorithms.

If the following error is returned by WebSphere MQ Advanced Message Security:

```
DRQJP0103E   The IBM WebSphere MQ Advanced Message Security Java interceptor failed to protect message.
  com.ibm.security.pkcsutil.PKCSException: Error encrypting contents (java.security.InvalidKeyException: Illegal key siz
```

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 correct Java policy file from the following locations:

- IBM SDK Policy files for Java 1.4.2.
- IBM SDK Policy files for Java 5.0.
- IBM SDK Policy files for Java 6.0.
- IBM SDK Policy files for Java 7.0.

OSGi support

To use OSGi bundle with IBM WebSphere 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: WebSphere 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 WebSphere 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 WebSphere MQ Advanced Message Security as described in "Quick Start Guide for Java clients" on page 440.

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 430.

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 WebSphere 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 WebSphere MQ Advanced Message Security.

See "Message Channel Agent (MCA) interception" on page 452 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.

Monitoring and performance

A number of monitoring techniques are available in IBM WebSphere 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.

Depending on the size and complexity of your queue manager network, you can obtain a range of information from monitoring 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.

Related information:

Configuring

Administering WebSphere 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.

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 WebSphere 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 482
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 WebSphere 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 78 on page 481 illustrates the concept of instrumentation events.

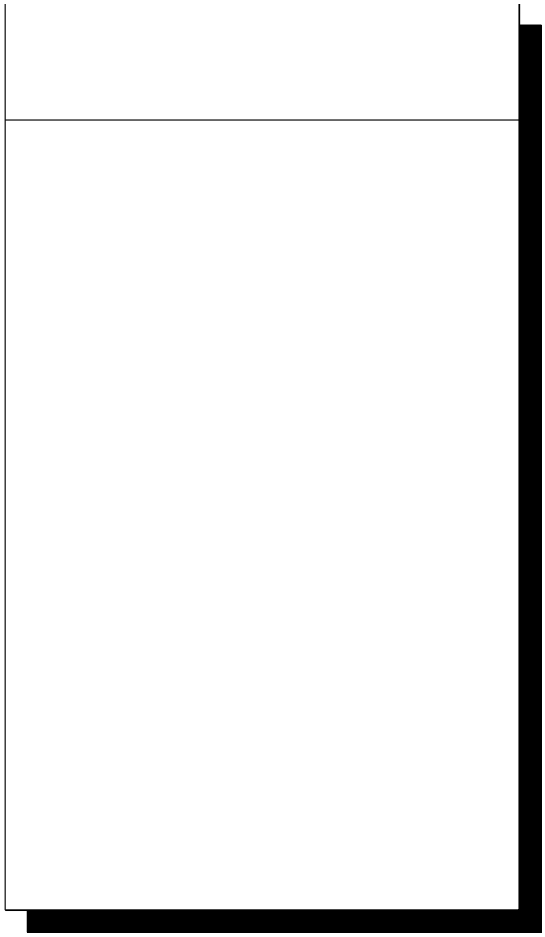


Figure 78. Understanding instrumentation events

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 WebSphere 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 493
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 523
Use this page to view a sample C program for monitoring instrumentation events

Event types

Use this page to view the types of instrumentation event that a queue manager or channel instance can report

IBM WebSphere 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" on page 154.

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

WebSphere MQ applications. In particular, you can monitor all the nodes in your system from a single node (for those nodes that support IBM WebSphere MQ events) as shown in Figure 79.

Instrumentation events can be reported through a user-written reporting mechanism to an administration application that can present the events to an operator.

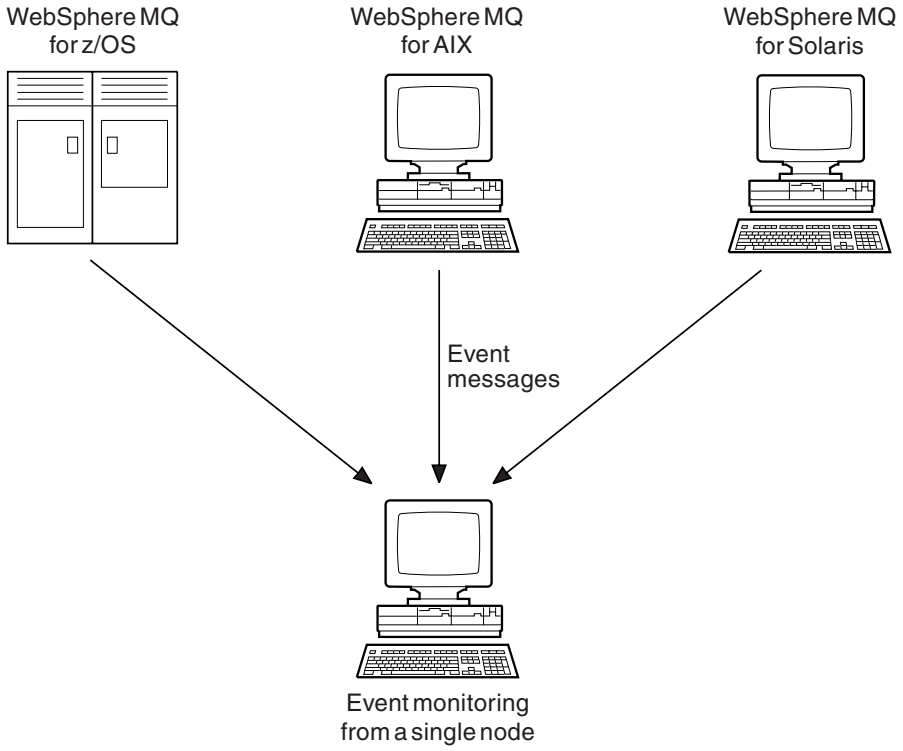


Figure 79. Monitoring queue managers across different platforms, on a single node

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.

WebSphere 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

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.

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.

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

Event type	See these topics
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
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 WebSphere MQ script commands (MQSC).
- The corresponding IBM WebSphere MQ PCF commands.

- The IBM WebSphere 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 480
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 482
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
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 WebSphere MQ objects.

Using Programmable Command Formats
You can use PCFs in a systems management program for WebSphere MQ remote administration.

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 32 summarizes how to enable queue manager events. To disable a queue manager event, set the appropriate parameter to `DISABLED`.

Table 32. 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 33 on page 490 summarizes how you enable channel and bridge events. To disable a queue manager event, set the appropriate parameter to `DISABLED`.

Table 33. 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, or as local definitions of remote 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.

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 488
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 WebSphere MQ messages, an event message has two parts: a message descriptor and the message data.

Related information:

QSGDisp (MQLONG)

Conditions for a trigger event

Format of event messages

Event messages contain information about an event and its cause. Like other WebSphere 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 575
Use this page to view the values contained by the MQMD structure for an activity report

"Activity report MQEPH (Embedded PCF header)" on page 579
Use this page to view the values contained by the MQEPH structure for an activity report

"Activity report MQCFH (PCF header)" on page 580
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 WebSphere MQ commands (MQSC). On , you can also set queue attributes using the operations and controls panels for queue managers.

Related reference:

"Event types" on page 482
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 34 on page 494 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 34. 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.

Related concepts:

"Performance events" on page 493
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 496
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 496
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 497
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

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 80 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 80. 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 496.

Related information:

Queue Service Interval OK

Queue Service Interval High

QServiceIntervalEvent (MQLONG)

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 35. 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 these examples to understand the information that you can obtain from queue service interval events

The three examples provide progressively more complex illustrations of the use of queue service interval events.

The figures accompanying the examples 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 494
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 496
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.

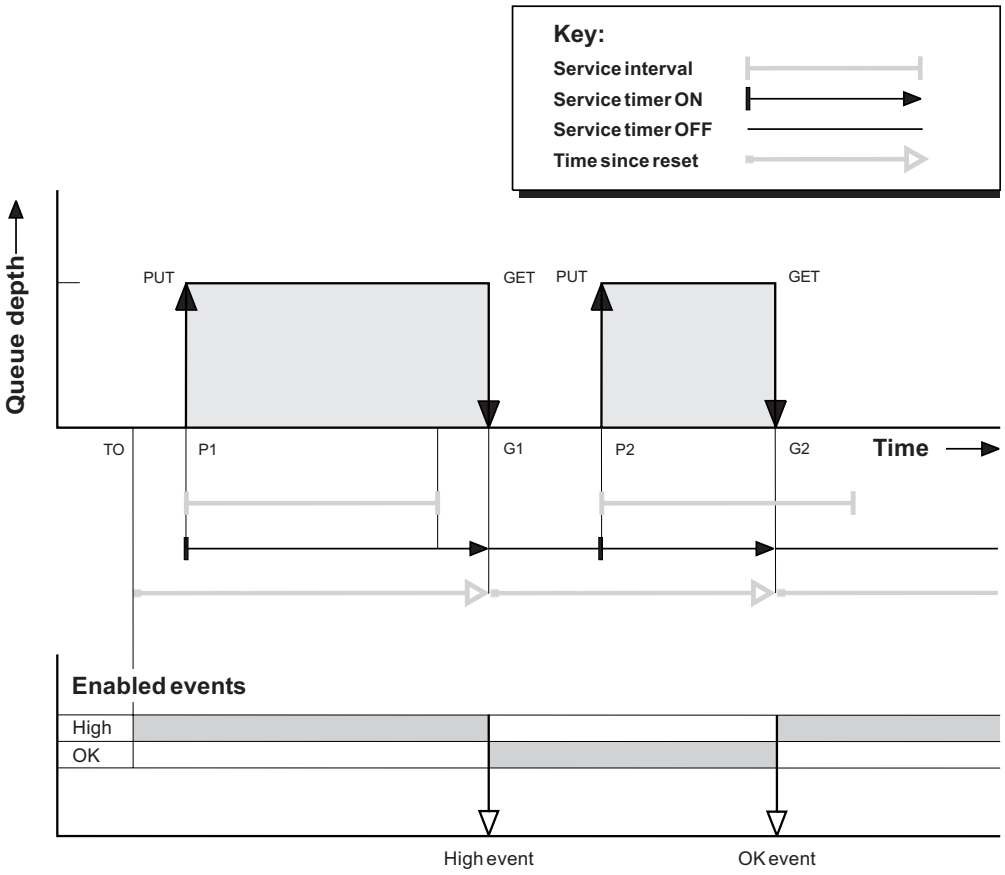


Figure 81. Queue service interval events - example 1

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 36 summarizes the event statistics for this example.

Table 36. 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 81 on page 499 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.

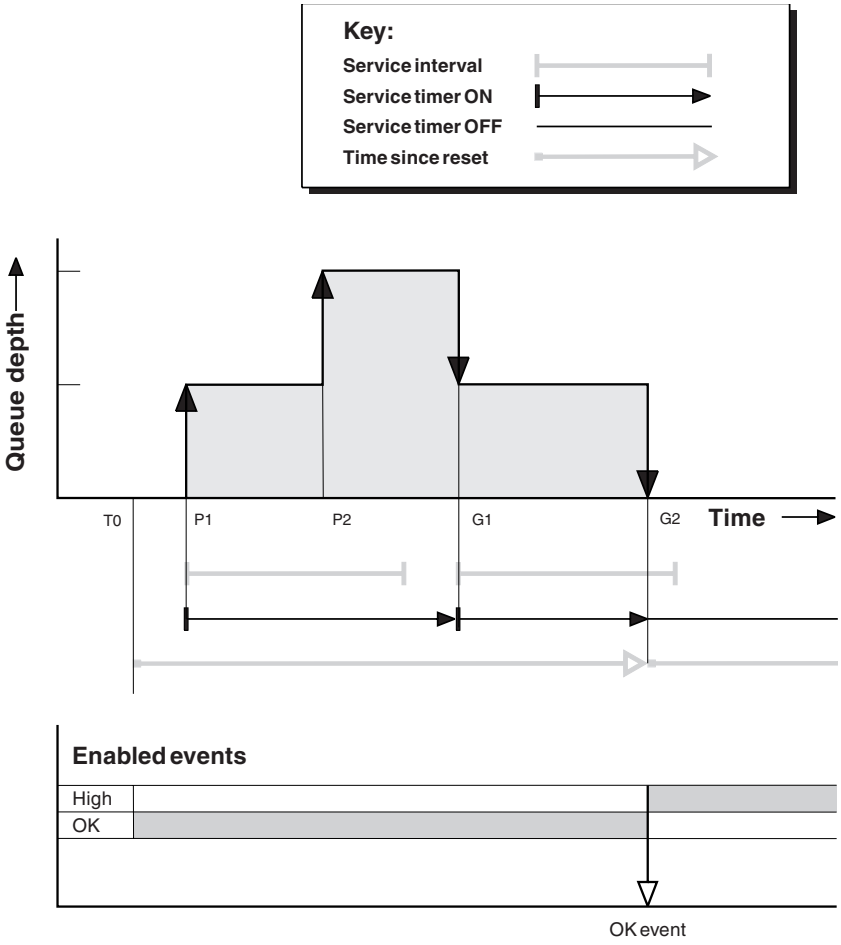


Figure 82. Queue service interval events - example 2

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 37 on page 502 summarizes the event statistics for this example.

Table 37. 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.

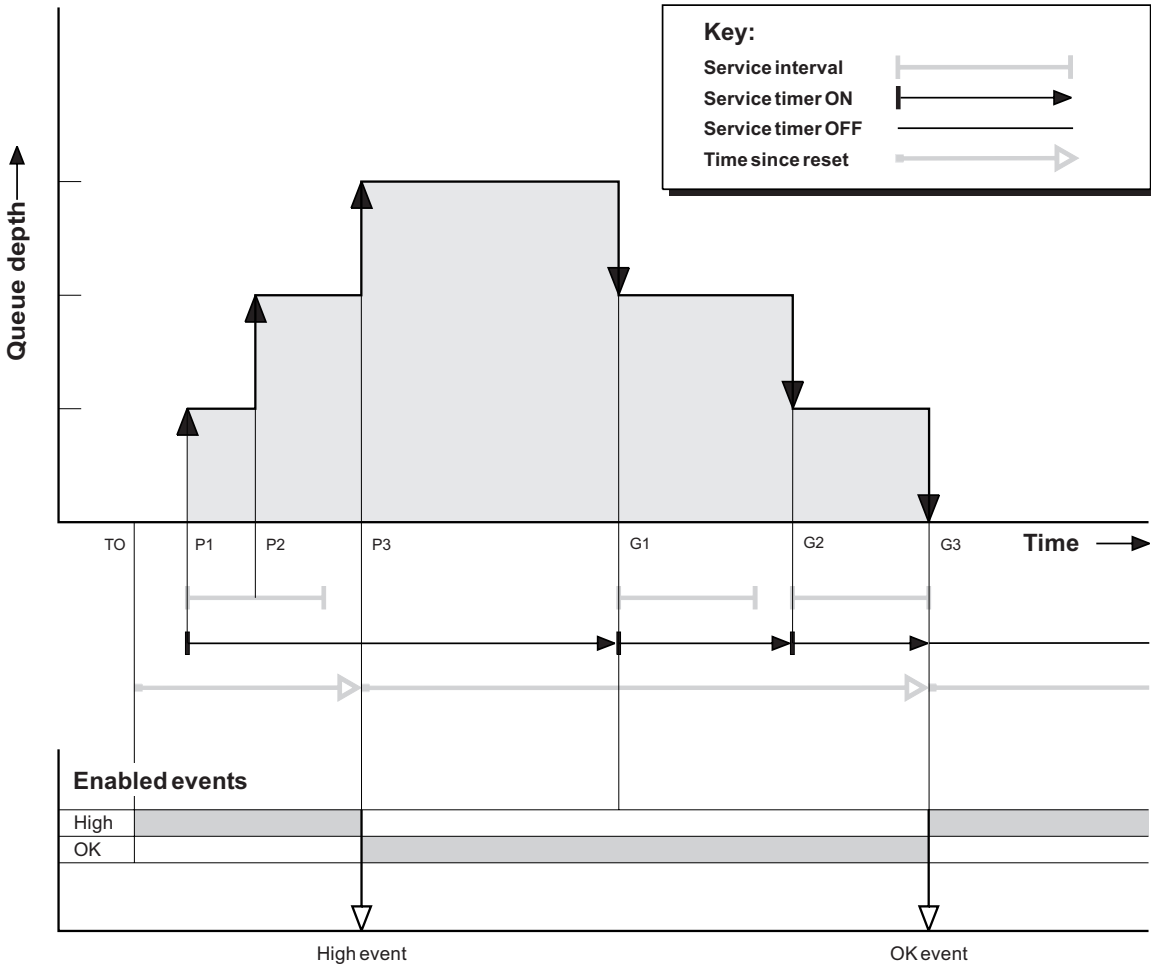


Figure 83. Queue service interval events - example 3

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 38 summarizes the event statistics for this example.

Table 38. 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 WebSphere 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, WebSphere 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 506
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 WebSphere 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)
```

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 WebSphere 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 503
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 504
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 84 on page 507 shows the variation of queue depth over time.

Figure 84. 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 39 on page 508 summarizes the queue event statistics and Table 40 on page 508 summarizes which events are enabled.

Table 39. 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 40. 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 85 on page 509 shows the variation of queue depth over time.

Figure 85. 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 41 on page 510 summarizes the queue event statistics and Table 42 on page 510 summarizes which events are enabled at different times for this example.

Table 41. 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 42. 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 486
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 482
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 510
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
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.

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 510
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
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.

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 510
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 482
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

When 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 514
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 514
Command events are notifications that an MQSC, or PCF command has run successfully.

"Command event generation" on page 515
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.

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 482
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 516
Logger events are notifications that a queue manager has started writing to a new log extent.

Related information:

LoggerEvent (MQLONG)

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 516
Logger events are notifications that a queue manager has started writing to a new log extent.

Related reference:

"Logger event generation" on page 516
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)

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 517
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 515
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 516
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

Sample program to monitor instrumentation events

Use this page to view a sample C program for monitoring instrumentation events

This sample program is not part of any IBM WebSphere MQ product and is therefore not supplied as an actual physical item. The example is incomplete in that it does not enumerate all the possible outcomes of specified actions. However, you can use this sample as a basis for your own programs that use events, in particular, the PCF formats used in event messages. However, you need to modify this program before running it on your own systems.

```
/********************************************************************/
/*                                                                  */
/* Program name: EVMON                                              */
/*                                                                  */
/* Description: C program that acts as an event monitor             */
/*                                                                  */
/*                                                                  */
/********************************************************************/
/*                                                                  */
/* Function:                                                        */
/*                                                                  */
/*                                                                  */
/*   EVMON is a C program that acts as an event monitor - reads an  */
/*   event queue and tells you if anything appears on it            */
/*                                                                  */
/*   Its first parameter is the queue manager name, the second is   */
/*   the event queue name. If these are not supplied it uses the    */
/*   defaults.                                                      */
/*                                                                  */
/********************************************************************/
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef min
  #define min(a,b)        (((a) < (b)) ? (a) : (b))
#endif

/********************************************************************/
/* includes for MQI                                                 */
/********************************************************************/
#include <cmqc.h>
#include <cmqcfc.h>
void printfmqcfst(MQCFST* pmqcfst);
void printfmqcfin(MQCFIN* pmqcfst);
void printreas(MQLONG reason);

 #define PRINTREAS(param)                                          \
```

```
    case param:                                                    \
      printf("Reason = %s\n",#param);                              \
      break;


/****************************************************************/
/* global variable                                              */
/****************************************************************/
MQCFH    *evtmsg;                    /* evtmsg message buffer       */

int main(int argc, char **argv)
{
  /**************************************************************/
  /* declare variables                                          */
  /**************************************************************/
  int  i;                             /* auxiliary counter          */
  /**************************************************************/
  /* Declare MQI structures needed                              */
  /**************************************************************/
  MQOD     od = {MQOD_DEFAULT};      /* Object Descriptor          */
  MQMD     md = {MQMD_DEFAULT};      /* Message Descriptor         */
  MQGMO   gmo = {MQGMO_DEFAULT};     /* get message options        */
  /**************************************************************/
  /* note, uses defaults where it can                           */
  /**************************************************************/

  MQHCONN  Hcon;                     /* connection handle          */
  MQHOBJ   Hobj;                     /* object handle              */
  MQLONG   O_options;                /* MQOPEN options             */
  MQLONG   C_options;                /* MQCLOSE options            */
  MQLONG   CompCode;                 /* completion code            */
  MQLONG   OpenCode;                 /* MQOPEN completion code     */
  MQLONG   Reason;                   /* reason code                */
  MQLONG   CReason;                  /* reason code for MQCONN     */
  MQLONG   buflen;                   /* buffer length              */
  MQLONG   evtmsglen;                /* message length received    */
  MQCHAR   command[1100];            /* call command string ...    */
  MQCHAR   p1[600];                  /* ApplId insert              */
  MQCHAR   p2[900];                  /* evtmsg insert              */
  MQCHAR   p3[600];                  /* Environment insert         */
  MQLONG   mytype;                   /* saved application type     */
  char     QMName[50];               /* queue manager name         */
  MQCFST  *paras;                    /* the parameters             */
  int      counter;                  /* loop counter               */
  time_t   ltime;

  /**************************************************************/
  /* Connect to queue manager                                   */
  /**************************************************************/
  QMName[0] = 0;                      /* default queue manager     */
  if (argc > 1)
    strcpy(QMName, argv[1]);
  MQCONN(QMName,                      /* queue manager             */
         &Hcon,                   /* connection handle         */
         &CompCode,               /* completion code           */
         &CReason);               /* reason code               */



  /**************************************************************/
  /* Initialize object descriptor for subject queue             */
  /**************************************************************/
  strcpy(od.ObjectName, "SYSTEM.ADMIN.QMGR.EVENT");
  if (argc > 2)
    strcpy(od.ObjectName, argv[2]);
```

```
/*****************************************************************/
/* Open the event queue for input; exclusive or shared.  Use of  */
/* the queue is controlled by the queue definition here          */
/*****************************************************************/

O_options = MQOO_INPUT_AS_Q_DEF     /* open queue for input       */
      + MQOO_FAIL_IF_QUIESCING      /* but not if qmgr stopping   */
      + MQOO_BROWSE;
MQOPEN(Hcon,                        /* connection handle          */
       &od,                    /* object descriptor for queue*/
       O_options,                   /* open options               */
       &Hobj,                  /* object handle              */
       &CompCode,              /* completion code            */
       &Reason);               /* reason code                */

/*****************************************************************/
/*   Get messages from the message queue                         */
/*****************************************************************/
while (CompCode != MQCC_FAILED)
{
  /***************************************************************/
  /* I don't know how big this message is so just get the        */
  /* descriptor first                                            */
  /***************************************************************/
  gmo.Options = MQGMO_WAIT + MQGMO_LOCK
     + MQGMO_BROWSE_FIRST + MQGMO_ACCEPT_TRUNCATED_MSG;
                                  /* wait for new messages      */
  gmo.WaitInterval = MQWI_UNLIMITED;/* no time limit             */
  buflen = 0;                     /* amount of message to get   */

  /***************************************************************/
  /* clear selectors to get messages in sequence                 */
  /***************************************************************/
  memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
  memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));

  /***************************************************************/
  /* wait for event message                                      */
  /***************************************************************/
  printf("...>\n");
  MQGET(Hcon,                         /* connection handle          */
        Hobj,                         /* object handle              */
        &md,                    /* message descriptor         */
        &gmo,                   /* get message options        */
        buflen,                       /* buffer length              */
        evtmsg,                       /* evtmsg message buffer      */
        &evtmsglen,             /* message length             */
        &CompCode,              /* completion code            */
        &Reason);               /* reason code                */



  /***************************************************************/
  /* report reason, if any                                       */
  /***************************************************************/
   if (Reason != MQRC_NONE && Reason != MQRC_TRUNCATED_MSG_ACCEPTED)
  {
    printf("MQGET ==> %ld\n", Reason);
  }
  else
  {
    gmo.Options = MQGMO_NO_WAIT + MQGMO_MSG_UNDER_CURSOR;
    buflen = evtmsglen;           /* amount of message to get   */
    evtmsg = malloc(buflen);
    if (evtmsg != NULL)
    {
      /*************************************************************/
```

```
    /* clear selectors to get messages in sequence              */
    /*************************************************************/
    memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));

    /*************************************************************/
    /* get the event message                                     */
    /*************************************************************/
    printf("...>\n");
    MQGET(Hcon,                     /* connection handle        */
          Hobj,                     /* object handle            */
          &md,                 /* message descriptor       */
          &gmo,                /* get message options      */
          buflen,                   /* buffer length            */
          evtmsg,                   /* evtmsg message buffer    */
          &evtmsglen,          /* message length           */
          &CompCode,           /* completion code          */
          &Reason);            /* reason code              */

    /*************************************************************/
    /* report reason, if any                                     */
    /*************************************************************/
    if (Reason != MQRC_NONE)
    {
      printf("MQGET ==> %ld\n", Reason);
    }
  }
  else
  {
    CompCode = MQCC_FAILED;
  }
}


/*****************************************************************/
/* . . . process each message received                           */
/*****************************************************************/

if (CompCode != MQCC_FAILED)
{
  /***************************************************************/
  /* announce a message                                          */
  /***************************************************************/
  printf("\a\a\a\a\a\a\a");
  time(&ltime);
  printf(ctime(&ltime));

  if (evtmsglen != buflen)
    printf("DataLength = %ld?\n", evtmsglen);
  else
  {
    /*************************************************************/
    /* right let's look at the data                              */
    /*************************************************************/
    if (evtmsg->Type != MQCFT_EVENT)
    {
      printf("Something's wrong this isn't an event message,"
             " its type is %ld\n",evtmsg->Type);
    }
    else
    {
      if (evtmsg->Command == MQCMD_Q_MGR_EVENT)
      {
        printf("Queue Manager event: ");
      }
      else
        if (evtmsg->Command == MQCMD_CHANNEL_EVENT)
```

```
  {
    printf("Channel event: ");
  }
  else

:


  {
    printf("Unknown Event message, %ld.",
            evtmsg->Command);
  }

if      (evtmsg->CompCode == MQCC_OK)
  printf("CompCode(OK)\n");
else if (evtmsg->CompCode == MQCC_WARNING)
  printf("CompCode(WARNING)\n");
else if (evtmsg->CompCode == MQCC_FAILED)
  printf("CompCode(FAILED)\n");
else
  printf("* CompCode wrong * (%ld)\n",
            evtmsg->CompCode);

if (evtmsg->StrucLength != MQCFH_STRUC_LENGTH)
{
  printf("it's the wrong length, %ld\n",evtmsg->StrucLength);
}

if (evtmsg->Version != MQCFH_VERSION_1)
{
  printf("it's the wrong version, %ld\n",evtmsg->Version);
}

if (evtmsg->MsgSeqNumber != 1)
{
  printf("it's the wrong sequence number, %ld\n",
          evtmsg->MsgSeqNumber);
}

if (evtmsg->Control != MQCFC_LAST)
{
  printf("it's the wrong control option, %ld\n",
          evtmsg->Control);
}

printreas(evtmsg->Reason);
printf("parameter count is %ld\n", evtmsg->ParameterCount);
/*********************************************************/
/* get a pointer to the start of the parameters          */
/*********************************************************/


paras = (MQCFST *)(evtmsg + 1);
counter = 1;
while (counter <= evtmsg->ParameterCount)
{
  switch (paras->Type)
  {
    case MQCFT_STRING:
      printfmqcfst(paras);
      paras = (MQCFST *)((char *)paras
                         + paras->StrucLength);
      break;
    case MQCFT_INTEGER:
      printfmqcfin((MQCFIN*)paras);
      paras = (MQCFST *)((char *)paras
                         + paras->StrucLength);
```

```
              break;
            default:
              printf("unknown parameter type, %ld\n",
                     paras->Type);
              counter = evtmsg->ParameterCount;
              break;
           }
           counter++;
         }
       }
     }   /* end evtmsg action             */
     free(evtmsg);
     evtmsg = NULL;
   }     /* end process for successful GET */
 }       /* end message processing loop    */

 /*****************************************************************/
 /* close the event queue - if it was opened                      */
 /*****************************************************************/
 if (OpenCode != MQCC_FAILED)
 {
   C_options = 0;                  /* no close options             */
   MQCLOSE(Hcon,                   /* connection handle            */
          &Hobj,               /* object handle               */
          C_options,
          &CompCode,           /* completion code             */
          &Reason);            /* reason code                 */
 /*****************************************************************/
 /* Disconnect from queue manager (unless previously connected)   */
 /*****************************************************************/
 if (CReason != MQRC_ALREADY_CONNECTED)
 {
   MQDISC(&Hcon,               /* connection handle           */
          &CompCode,           /* completion code             */
          &Reason);            /* reason code                 */


/*****************************************************************/
/*                                                               */
/* END OF EVMON                                                  */
/*                                                               */
/*****************************************************************/
}

#define PRINTPARAM(param)                                        \
   case param:                                                   \
     {                                                           \
       char *p = #param;                                         \
     strncpy(thestring,pmqcfst->String,min(sizeof(thestring),    \
             pmqcfst->StringLength));                            \
     printf("%s %s\n",p,thestring);                              \
     }                                                           \
     break;

#define PRINTAT(param)                                           \
   case param:                                                   \
     printf("MQIA_APPL_TYPE = %s\n",#param);                     \
     break;


void printfmqcfst(MQCFST* pmqcfst)
{
  char thestring[100];

  switch (pmqcfst->Parameter)
  {
    PRINTPARAM(MQCA_BASE_Q_NAME)
```

```
    PRINTPARAM(MQCA_PROCESS_NAME)
    PRINTPARAM(MQCA_Q_MGR_NAME)
    PRINTPARAM(MQCA_Q_NAME)
    PRINTPARAM(MQCA_XMIT_Q_NAME)
    PRINTPARAM(MQCACF_APPL_NAME)

.
.
.
     default:
      printf("Invalid parameter, %ld\n",pmqcfst->Parameter);
      break;
  }
}



void printfmqcfin(MQCFIN* pmqcfst)
{
  switch (pmqcfst->Parameter)
  {
    case MQIA_APPL_TYPE:
      switch (pmqcfst->Value)
      {
        PRINTAT(MQAT_UNKNOWN)
        PRINTAT(MQAT_OS2)
        PRINTAT(MQAT_DOS)
        PRINTAT(MQAT_UNIX)
        PRINTAT(MQAT_QMGR)
        PRINTAT(MQAT_OS400)
        PRINTAT(MQAT_WINDOWS)
        PRINTAT(MQAT_CICS_VSE)
        PRINTAT(MQAT_VMS)
        PRINTAT(MQAT_GUARDIAN)
        PRINTAT(MQAT_VOS)
      }
      break;
    case MQIA_Q_TYPE:
      if (pmqcfst->Value == MQQT_ALIAS)
      {
        printf("MQIA_Q_TYPE is MQQT_ALIAS\n");
      }
      else

.
.
.
{
        if (pmqcfst->Value == MQQT_REMOTE)
        {
          printf("MQIA_Q_TYPE is MQQT_REMOTE\n");
          if (evtmsg->Reason == MQRC_ALIAS_BASE_Q_TYPE_ERROR)
          {
            printf("but remote is not valid here\n");
          }
        }
        else
        {
          printf("MQIA_Q_TYPE is wrong, %ld\n",pmqcfst->Value);
        }
      }
      break;


          case MQIACF_REASON_QUALIFIER:
      printf("MQIACF_REASON_QUALIFIER %ld\n",pmqcfst->Value);
      break;

    case MQIACF_ERROR_IDENTIFIER:
      printf("MQIACF_ERROR_INDENTIFIER %ld (X'%lX')\n",
```

```
              pmqcfst->Value,pmqcfst->Value);
      break;

    case MQIACF_AUX_ERROR_DATA_INT_1:
      printf("MQIACF_AUX_ERROR_DATA_INT_1 %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;

    case MQIACF_AUX_ERROR_DATA_INT_2:
      printf("MQIACF_AUX_ERROR_DATA_INT_2 %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;
:
default :
      printf("Invalid parameter, %ld\n",pmqcfst->Parameter);
      break;
  }
}

   void printreas(MQLONG reason)
{
  switch (reason)
  {
    PRINTREAS(MQRCCF_CFH_TYPE_ERROR)
    PRINTREAS(MQRCCF_CFH_LENGTH_ERROR)
    PRINTREAS(MQRCCF_CFH_VERSION_ERROR)
    PRINTREAS(MQRCCF_CFH_MSG_SEQ_NUMBER_ERR)

:
    PRINTREAS(MQRC_NO_MSG_LOCKED)
    PRINTREAS(MQRC_CONNECTION_NOT_AUTHORIZED)
    PRINTREAS(MQRC_MSG_TOO_BIG_FOR_CHANNEL)
    PRINTREAS(MQRC_CALL_IN_PROGRESS)
    default:
      printf("It's an unknown reason, %ld\n",
              reason);
      break;
  }
}
```

Related concepts:

"Instrumentation events" on page 480
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.

"Event monitoring" on page 479
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.

Related reference:

"Sample program to monitor the logger event queue" on page 517
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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 531
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 540
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 WebSphere MQ Version 6.0 and subsequent releases
MCAs and user-written applications connected to queue managers at 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 WebSphere MQ Version 6.0 or subsequent releases only, complete message routes can be determined.

WebSphere MQ queue managers before Version 6.0
Applications connected to WebSphere 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 WebSphere MQ queue manager prior to Version 6.0, only a partial message route can be determined.

How activity information is stored

WebSphere 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 540
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 546
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 582
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 WebSphere 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 WebSphere 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 WebSphere 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 a WebSphere 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 WebSphere 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 598
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 WebSphere 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 WebSphere 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 536.

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 544
- 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 544
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 WebSphere MQ display route application.

Related tasks:

"Controlling activity recording" on page 536
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:

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 598.

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 WebSphere 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 WebSphere MQ display route application

Use the WebSphere 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 WebSphere MQ display route application.

Restriction: dspmqrte cannot be issued on queue managers before WebSphere MQ Version 6.0 or on WebSphere 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 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 WebSphere 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 WebSphere 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 551.

Table 43. 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:

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:

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 a WebSphere MQ Version 6 (or later) queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by a WebSphere 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:

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 44. Group name

Description	**Grouped parameters specifying the additional information.**
Identifier	MQGACF_VALUE_NAMING.
Data type	MQCFGR
Parameters in group	*ParameterName* *ParameterValue*

ParameterName

Table 45. Parameter name

Description	**Contains the name to be displayed by the IBM WebSphere 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 46. Parameter value

Description	**Contains the value to be displayed by the IBM WebSphere 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 WebSphere 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 WebSphere MQ display route application.

1. The WebSphere 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 WebSphere 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 WebSphere MQ display route application.
3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the WebSphere MQ display route application. The additional activity information is displayed as follows:

   ```
   65536: 'Red'
   ```

   The WebSphere 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 WebSphere 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 WebSphere MQ display route application.

1. TheIBM WebSphere 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.”
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 47. Color information

Description	**Grouped parameters specifying information about a color.**
Identifier:	MQGACF_VALUE_NAMING.
Data type:	MQCFGR.
Parameters in group:	*ColorName* *ColorValue*

ColorName

Table 48. Color name

Description	**Contains the name to be displayed by the IBM WebSphere 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 49. Color value

Description	**Contains the value to be displayed by the IBM WebSphere 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 WebSphere MQ display route application.

3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the IBM WebSphere MQ display route application. The additional activity information is displayed as follows:

   ```
   Color: 'Red'
   ```

   The IBM WebSphere 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.

WebSphere MQ display route application

Use the WebSphere 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.

Note: To run a Client Application against a queue manager, the Client Attachment feature must be installed.

You can use the WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ display route application connects to

-c

Specifies that the WebSphere MQ display route application connects as a client application.

If you do not specify this parameter, the WebSphere MQ display route application does not connect as a client application.

-m *QMgrName*

The name of the queue manager to which the WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ display route application to publish

-ts *TopicName*
Specifies a topic string to which the WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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:

The activity report option, specified using -ro
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 a IBM WebSphere MQ queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by a IBM WebSphere 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 WebSphere MQ display route application uses a temporary dynamic queue to store the returned messages. When the IBM WebSphere 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 WebSphere 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 WebSphere 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 546. The possible values for *Forward* are:

all The trace-route message is forwarded to any queue manager.

Warning: If forwarded to a IBM WebSphere 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.

The discard report option, specified using -ro
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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 551.

WebSphere MQ display route application examples

The following examples show how you can use the WebSphere 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 WebSphere 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 WebSphere MQ display route application using activity reports is put in order and displayed.

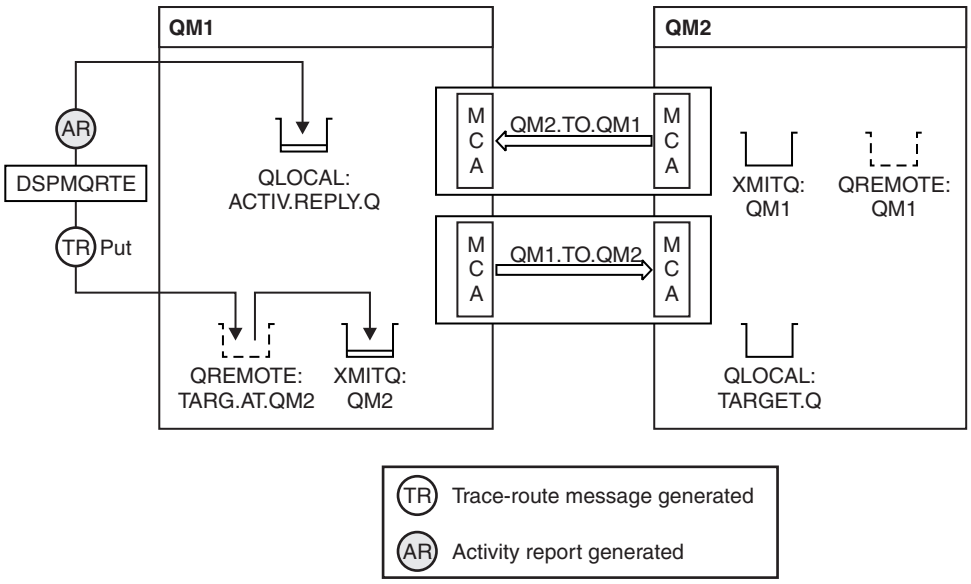


Figure 86. Requesting activity reports, Diagram 1

- 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 WebSphere 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.

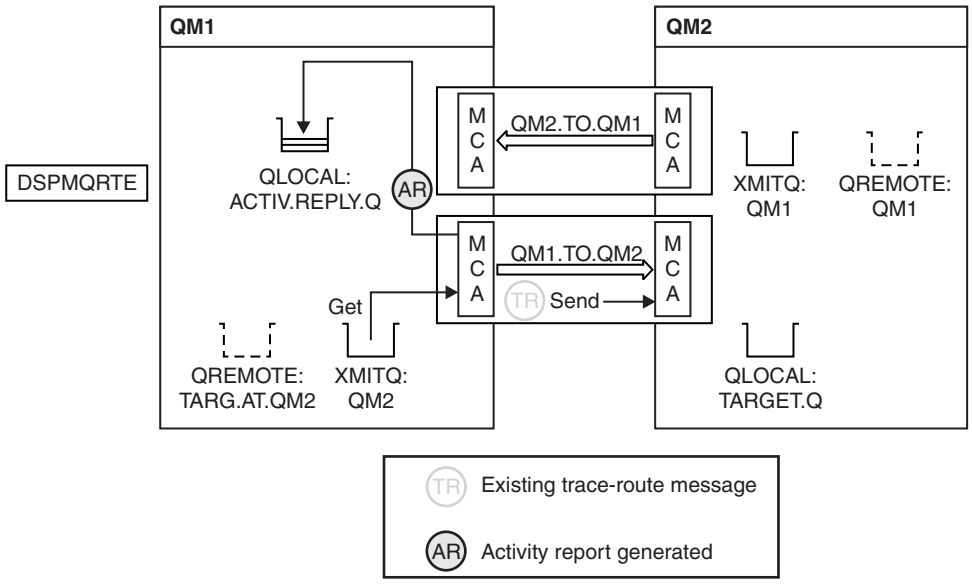


Figure 87. Requesting activity reports, Diagram 2

- 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 548, 4 on page 548, and 5 on page 548) 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).

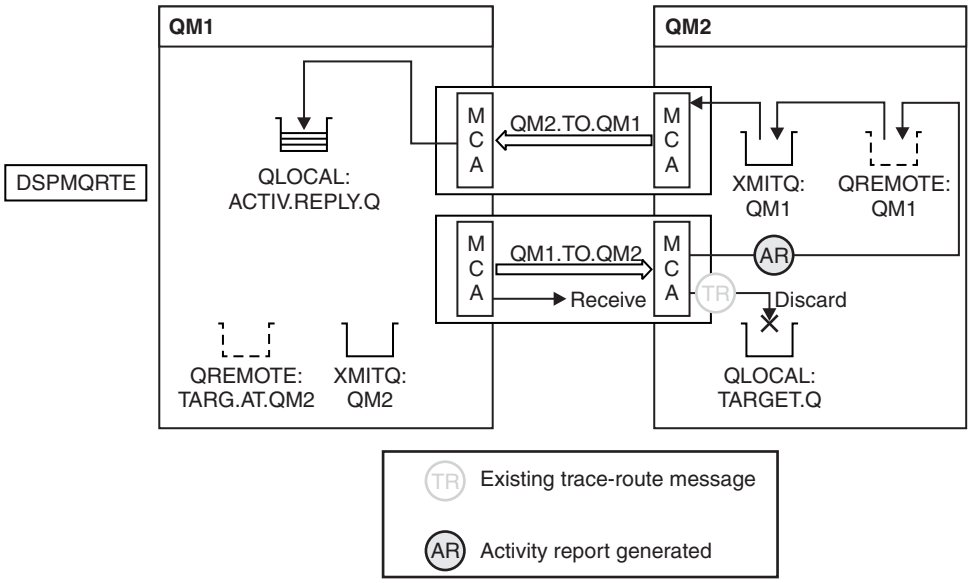


Figure 88. Requesting activity reports, Diagram 3

- 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).

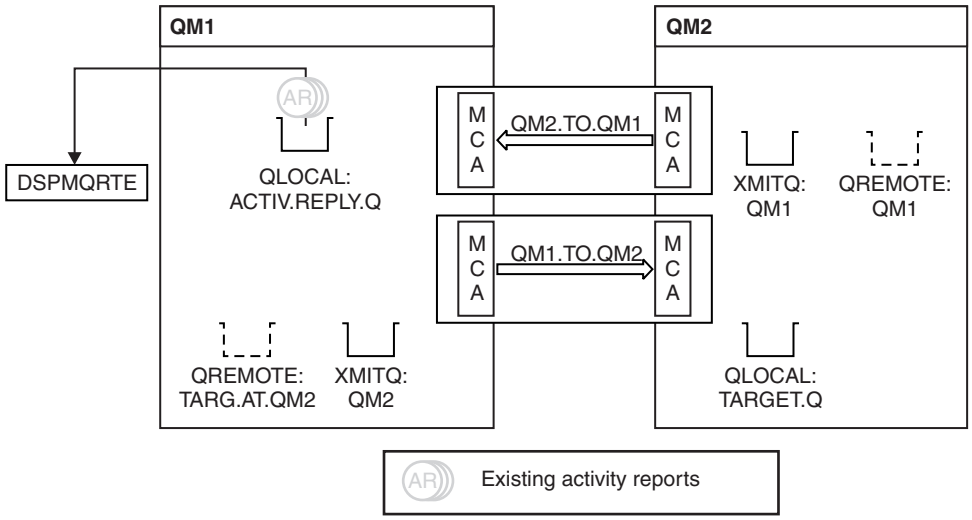


Figure 89. Requesting activity reports, Diagram 4

- 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 WebSphere 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.

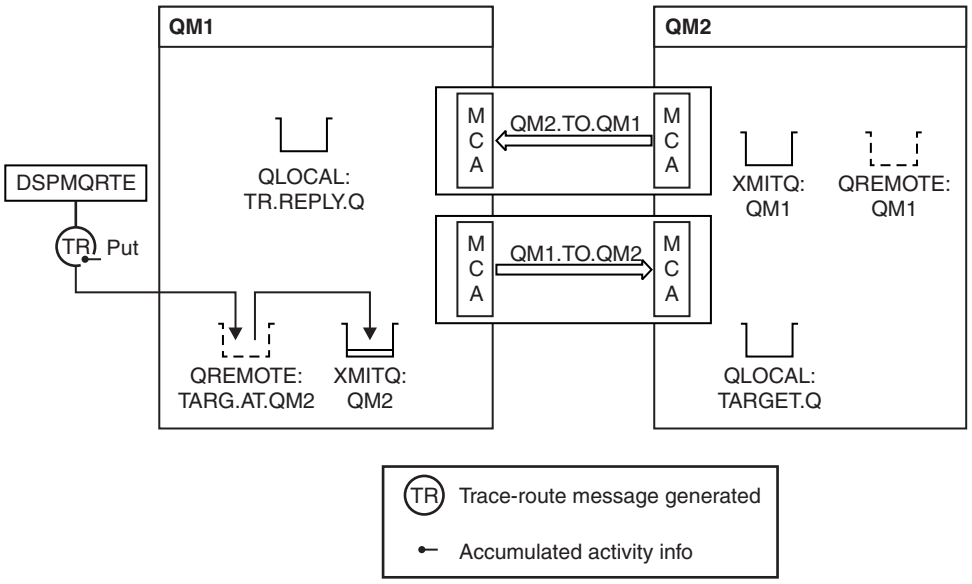


Figure 90. Requesting a trace-route reply message, Diagram 1

- 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 WebSphere 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.

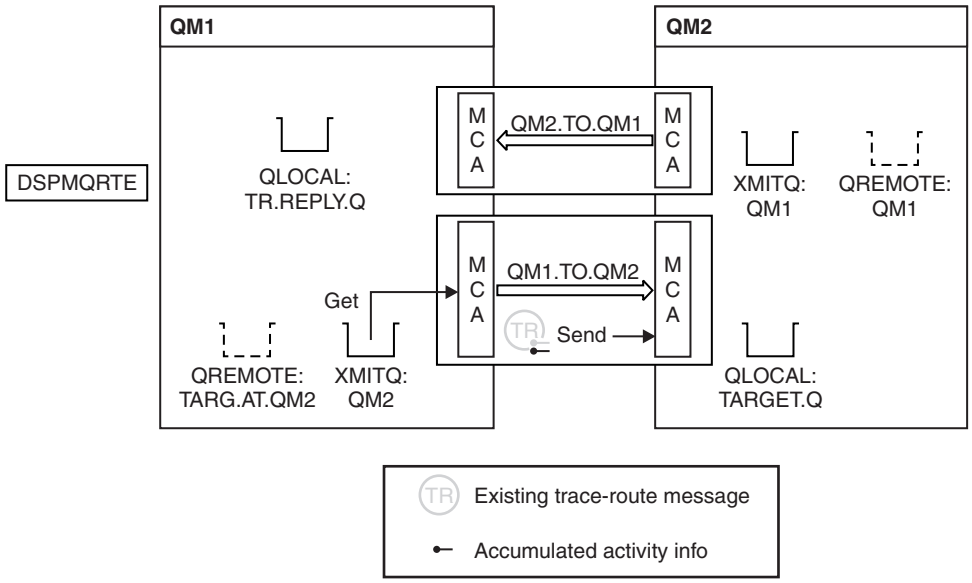


Figure 91. Requesting a trace-route reply message, Diagram 2

- 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.

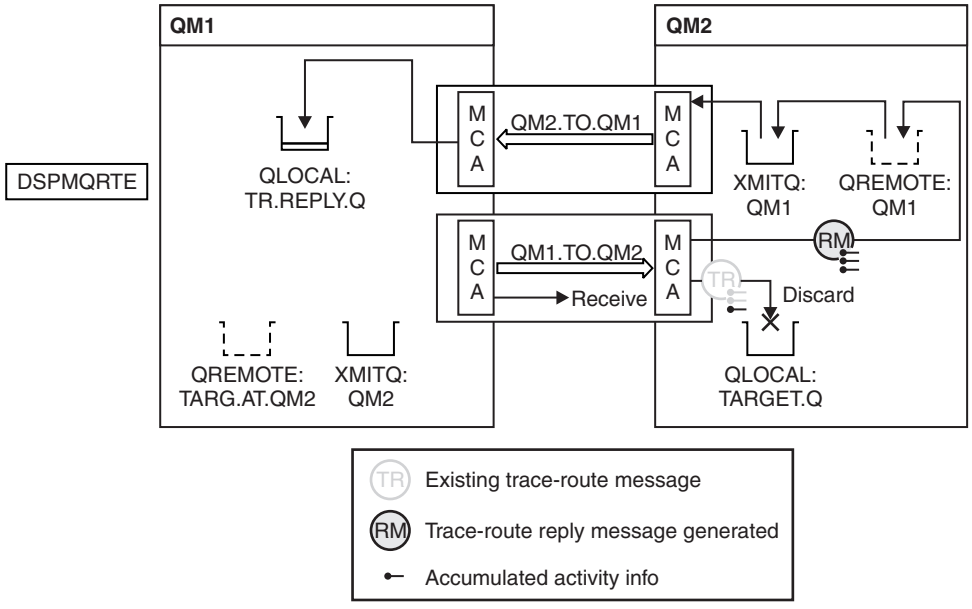


Figure 92. Requesting a trace-route reply message, Diagram 3

- 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.

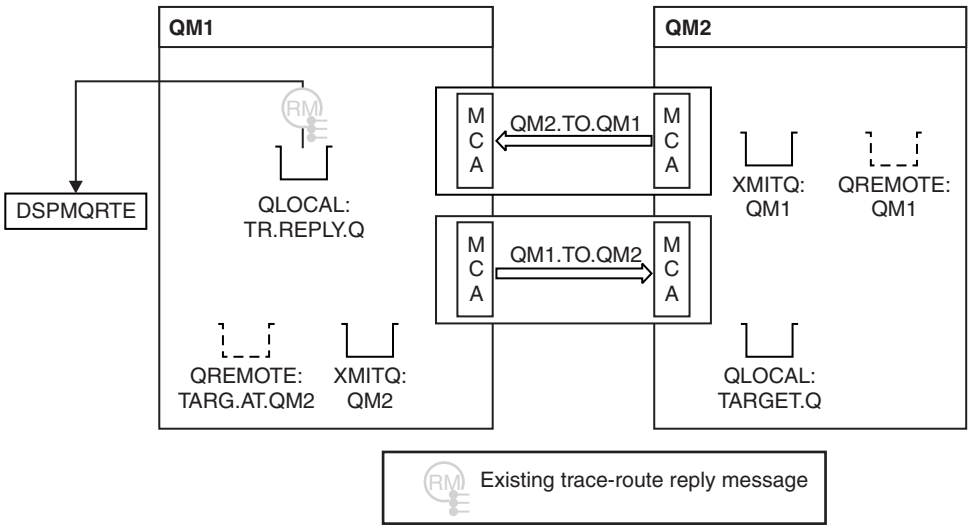


Figure 93. Requesting a trace-route reply message, Diagram 4

- 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 WebSphere 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 562, 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 WebSphere MQ display route application is used to read activity reports from another queue.

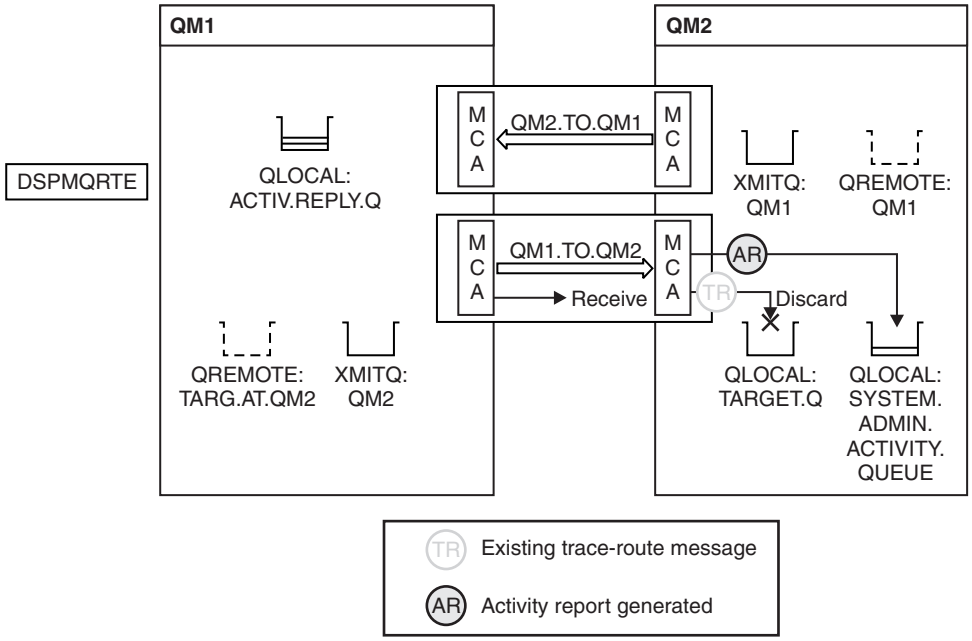


Figure 94. Delivering activity reports to the system queue, Diagram 1

- 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).

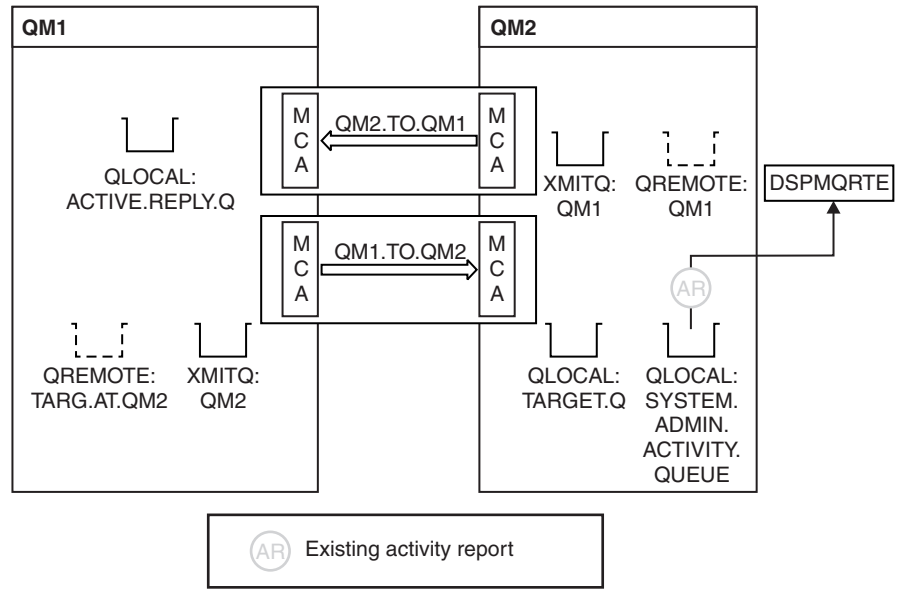


Figure 95. Delivering activity reports to the system queue, Diagram 2

- 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'414D51204C4152474551202020202020A3C9154220001502'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001503'
  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'414D51204C4152474551202020202020A3C9154220001505'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001502'

   EmbeddedMQMD:
    MsgId: X'414D51204C4152474551202020202020A3C9154220001502'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001503'
  QMgrName: 'QM1                                                '
  QName: 'QM2                                                  '
  ResolvedQName: 'QM2                                                  '

 Operation:
  OperationType: Send

  Message:

   MQMD:
    MsgId: X'414D51204C4152474551202020202020A3C9154220001502'
    CorrelId: X'414D51204C4152474551202020202020A3C9154220001503'
  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 414D51204C4152474551202020202020A3C9154220001502 -v outline
  ```

  where 414D51204C4152474551202020202020A3C9154220001502 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 414D51204C4152474551202020202020A3C915420001502 -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 WebSphere 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.

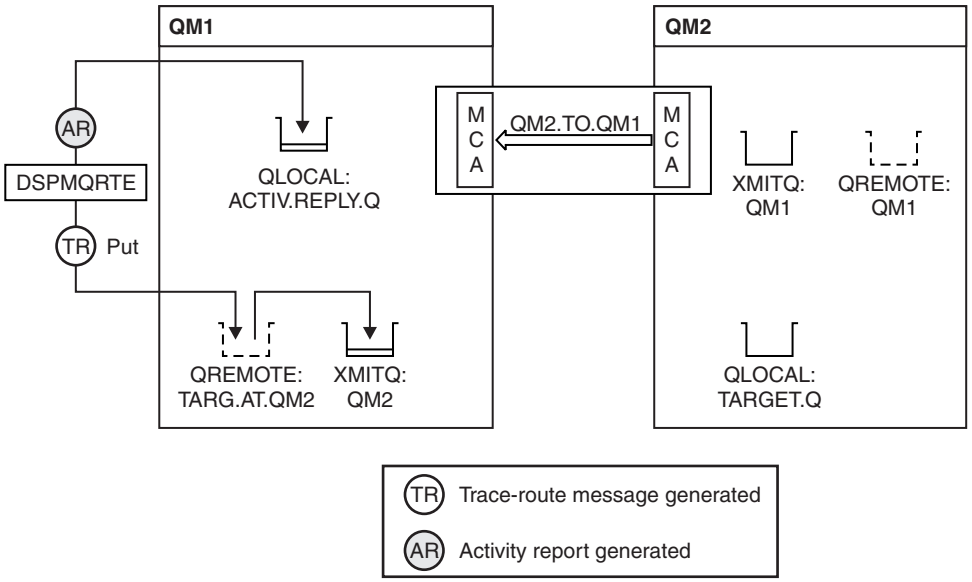


Figure 96. Diagnosing a channel problem

- 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 WebSphere 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 50 on page 574 shows the structure of these reports, including parameters that are returned only under certain conditions.

Table 50. Activity report format

MQMD structure	Embedded PCF header MQEPH structure	Activity report message data
Structure identifier Structure version Report options Message type Expiration time Feedback Encoding Coded character set ID Message format Priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier 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	Structure identifier Structure version Structure length Encoding Coded character set ID Message format Flags PCF header (MQCFH) Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	Activity Activity application name Activity application type Activity description Operation Operation type Operation date Operation time Message Message length MQMD [8] EmbeddedMQMD Queue manager name Queue sharing group name Queue name [1 2 3 7] Resolved queue name [1 3 7] Remote queue name [3 7] Remote queue manager name [2 3 4 5 7] Subscription level [9] Subscription identifier [9] Feedback [2 10] Channel name [4 5] Channel type [4 5] Transmission queue name [5] TraceRoute [6] Detail Recorded activities Unrecorded activities 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 WebSphere 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)."

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 a WebSphere 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 WebSphere 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 WebSphere 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 51 on page 600 shows the structure of a trace-route message.

Table 51. Trace-route message format

MQMD structure	Embedded PCF header MQEPH structure	Trace-route message data
Structure identifier Structure version Report options Message type Expiration time Feedback Encoding Coded character set ID Message format Priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier 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	Structure identifier Structure version Structure length Encoding Coded character set ID Message format Flags PCF header (MQCFH) Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	TraceRoute Detail Recorded activities Unrecorded activities Discontinuity count Max activities Accumulate Deliver

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 WebSphere 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 52 on page 610 shows the structure of a trace-route reply message, including parameters that are only returned under certain conditions.

Table 52. Trace-route reply message format

MQMD structure	PCF header MQCFH structure	Trace-route reply message data
Structure identifier Structure version Report options Message type Expiration time Feedback Encoding Coded character set ID Message format Priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier 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	PCF header (MQCFH) Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	Activity Activity application name Activity application type Activity description Operation Operation type Operation date Operation time Message Message length MQMD EmbeddedMQMD Queue manager name Queue sharing group name Queue name [1] [2] [3] Resolved queue name [1] [3] Remote queue name [3] Remote queue manager-name [2] [3] [4] [5] Feedback [2] Channel name [4] [5] Channel type [4] [5] Transmission queue name [5] TraceRoute Detail Recorded activities Unrecorded activities Discontinuity count Max activities 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 582.

Accounting and statistics messages

Queue managers generate accounting and statistics messages to record information about the MQI operations performed by IBM WebSphere MQ applications, or to record information about the activities occurring in a IBM WebSphere MQ system.

Accounting messages
Accounting messages are used to record information about the MQI operations performed by IBM WebSphere MQ applications, see "Accounting messages" on page 612.

Statistics messages
Statistics messages are used to record information about the activities occurring in a IBM WebSphere MQ system, see “Statistics messages” on page 615.

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.

Accounting messages

Accounting messages record information about the MQI operations performed by WebSphere 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 WebSphere 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 629
Use this page to view the structure of an MQI accounting message

“Queue accounting message data” on page 640
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 WebSphere 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 WebSphere 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 a WebSphere 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 617
Use the queue manager attribute STATMQI to control the collection of MQI statistics information

"Queue statistics information" on page 617
Use the queue attribute STATQ and the queue manager attribute STATQ to control the collection of queue statistics information

"Channel statistics information" on page 618
Use the channel attribute STATCHL to control the collection of channel statistics information. You can also set queue manager attributes to control information collection. These attributes are available on distributed platforms and on IBM i.

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 WebSphere 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:

Use the channel attribute STATCHL to control the collection of channel statistics information. You can also set queue manager attributes to control information collection. These attributes are available on distributed platforms and on IBM i.

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 statistics 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 statistics information collection is controlled at the queue manager level.

Automatically defined cluster-sender channels are not WebSphere 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 53 on page 619 summarizes the levels available with channel statistics information collection:

Table 53. 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.

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.

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 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.

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 WebSphere MQ that processes messages from the accounting and statistics queues and displays the information to the screen in a readable form.

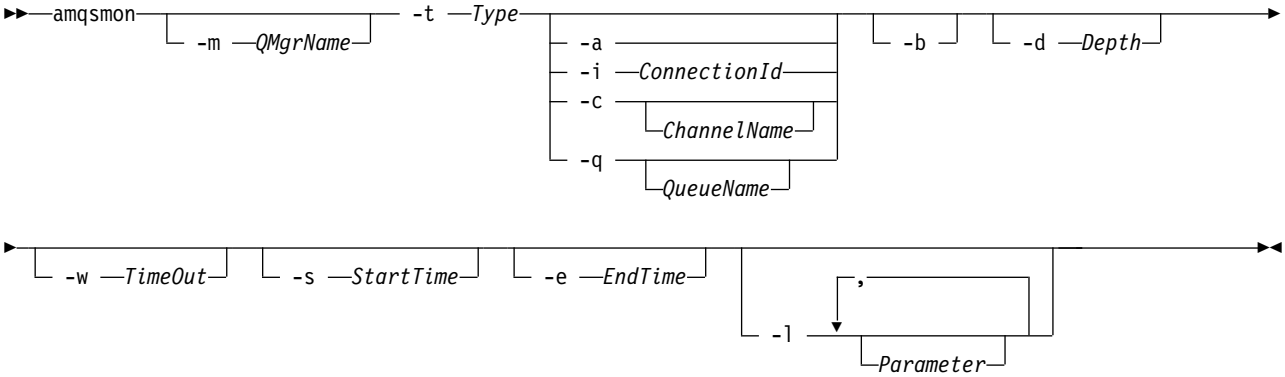
Because **amqsmon** is a sample program, you can use the supplied source code as template for writing your own application to process accounting or statistics messages, or modify the **amqsmon** source code to meet

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'414d514354524556312020202020208d0b3742010a0020'
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'414d514354524556312020202020208d0b3742010c0020'
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'414d514354524556312020202020208d0b374203090020'
ApplicationName: 'runmqsc'

ConnectionId: x'414d514354524556312020202020208d0b3742010a0020'
ApplicationName: 'amqsput'

ConnectionId: x'414d514354524556312020202020208d0b3742010c0020'
ApplicationName: 'runmqsc'

ConnectionId: x'414d514354524556312020202020208d0b3742010d0020'
ApplicationName: 'amqsput'

ConnectionId: x'414d514354524556312020202020208d0b3742150d0020'
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 WebSphere MQ messages containing a message descriptor and message data. The message data contains information about the MQI operations performed by WebSphere MQ applications, or information about the activities occurring in a WebSphere 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 54. MQI accounting message structure

MQMD structure	**Accounting message header MQCFH structure**	**MQI accounting message data** [1]
Structure identifier Structure version Report options Message type Expiration time Feedback code Encoding Coded character set ID Message format Message priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier 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	Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	Queue manager Interval start date Interval start time Interval end date Interval end time Command level Connection identifier Sequence number Application name Application process identifier Application thread identifier User identifier Connection date Connection time Connection name Channel name Disconnect date Disconnect time Disconnect type Open count Open fail count Close count Close fail count Put count Put fail count Put1 count Put1 fail count Put bytes 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 successful 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 WebSphere 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 WebSphere 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 WebSphere 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 629
- "Queue accounting message data" on page 640
- "MQI statistics message data" on page 650
- "Queue statistics message data" on page 661
- "Channel statistics message data" on page 668

1. This parameter relates to WebSphere MQ objects. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 55. 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 WebSphere MQ messages. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 56. 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 WebSphere 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.

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 682.

The simplest way to view the contents of application activity trace messages is to use the "amqsact sample program" on page 682.

Procedure

1. "Setting ACTVTRC to control collection of activity trace information."
2. "Setting MQCONNX options to control collection of activity trace information" on page 676.
3. "Configuring activity trace behavior using `mqat.ini`" on page 676.
4. "Tuning the performance impact of application activity trace" on page 682.

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 676.

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 682.

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 682.

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 682.

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 682.

Configuring activity trace behavior using mqat.ini

The activity trace behavior is configured using a configuration file called mqat.ini. This file follows the same stanza key and parameter-value pair format as the mqs.ini and qm.ini files.

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.

The syntax rules for the format of the file are:

- Text beginning with a hash or semi-colon is considered to be a comment which 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.

Stanza keys

Two stanza key types are allowed in the configuration file: the AllActivityTrace stanza, and the ApplicationTrace stanza

AllActivityTrace stanza

The AllActivityTrace stanza defines settings for the activity trace that is applied to all IBM WebSphere MQ connections unless overridden.

Individual values in the AllActivityTrace stanza can be overridden by more specific information in an ApplicationTrace 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

ApplicationTrace stanza

The ApplicationTrace stanza defines settings which can be applied to a specific name, type or both of IBM WebSphere MQ connection.

This stanza includes ApplName and ApplClass values which are used according to the matching rules defined in Connection Matching Rules to determine whether the stanza applies to a particular connection.

Parameter/Value Pairs

The following table lists the parameter/value pairs which may be used in the activity trace configuration file.

Table 57. Parameter/value pairs that can be used in the activity trace configuration file

Name	Stanza Type	Values (default in bold type)	Description
Trace	ApplicationTrace	ON / **OFF**	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.

Table 57. Parameter/value pairs that can be used in the activity trace configuration file (continued)

Name	Stanza Type	Values (default in bold type)	Description
ActivityInterval	AllActivityTrace ApplicationTrace	**0**-99999999 (**0=off**)	Time interval in seconds between trace messages. Activity trace does not use a timer thread, so the trace message will not be written at the exact instant that the time elapses - rather it will be written when the first MQI operation is executed after the time interval has elapsed. If this value is 0 then the trace message is written when the connection disconnects (or when the activity count is reached).
ActivityCount	AllActivityTrace ApplicationTrace	**0**-99999999 (**0=off**)	Number of MQI or XA operations between trace messages. If this value is 0 then the trace message is written when the connection disconnects (or when the activity interval has elapsed).
TraceLevel	AllActivityTrace ApplicationTrace	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	AllActivityTrace ApplicationTrace	**0**- 104 857 600 (100Mb)	Amount of message data traced in bytes for MQGET, MQPUT, MQPUT1, and Callback operations

Table 57. Parameter/value pairs that can be used in the activity trace configuration file (continued)

Name	Stanza Type	Values (default in bold type)	Description
ApplName	ApplicationTrace	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 distributed platforms, 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. 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 are. If the ApplName value is set to a single wildcard character (*) then the ApplName value matches all applications.
ApplClass	ApplicationTrace	**USER** / MCA / INTERNAL / ALL	The class of application. See the following table for an explanation of how the AppType values correspond to IBM WebSphere 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 58. 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)
INTERNAL	MQXACT_EXTERNAL	MQXE_COMMAND_SERVER MQXE_MQSC	'runmqsc' and command server
INTERNAL	MQXACT_INTERNAL	(Any value)	"trusted" and internal applications and processes; for example, amqzdmaa
ALL	(Any value)	(Any value)	All user and internal 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
```

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 WebSphere 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 connections ApplName. 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 stanzas ApplName value does not match the connection ApplName then the stanza is ignored for this connection.

4. If more than one stanza matches the connections ApplName and ApplClass, then the stanza with the most specific ApplName is used. The most specific ApplName is defined as the one which uses the most characters to match the connections ApplName. 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, then the values from the last matching will be used and all other stanzas will be ignored.

Application Activity Trace File Example

The following example shows how the configuration data is specified in the Activity Trace ini file. This example is shipped as a sample called `mqat.ini` in the C samples directory (the same directory as the `amqsact.c file`)

```
AllActivityTrace:
  ActivityInterval=0                  # Time interval between trace messages
                                      #   Values: 0-99999999 (0=off)
                                      #   Default: 0
  ActivityCount=0                     # 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=0                  # Amount of message data traced
                                      #   Values: 0-100000000
                                      #   Default: 0

ApplicationTrace:
  ApplClass=USER                       # Application type
                                      #   Values:  (USER | MCA | INTERNAL | ALL)
                                      #   Default: USER
  ApplName=AppName*                   # Application name (may be wildcarded)
                                      #   (matched to app name without path)
                                      #   Default: *
  Trace=OFF                           # Activity trace switch for application
                                      #   Values:  ( ON | OFF )
                                      #   Default: OFF
ActivityInterval=0                    # Time interval between trace messages
                                      #   Values: 0-99999999 (0=off)
                                      #   Default: 0
  ActivityCount=0                     # 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=0                  # Amount of message data traced
                                      #   Values: 0-100000000
                                      #   Default: 0
```

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 682.

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 676 and "Setting MQCONNX options to control collection of activity trace information" on page 676.
- 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 676.
- 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

amqsact sample program

amqsact formats Application Activity Trace messages for you and is provided with WebSphere MQ.

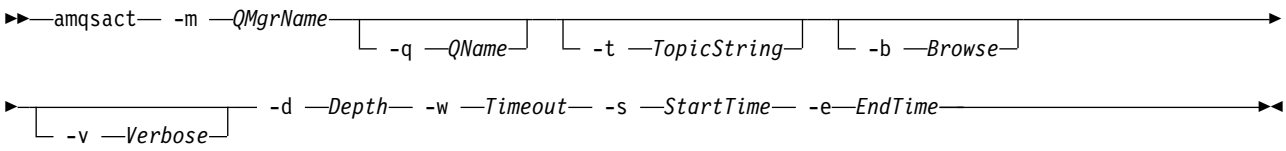
The compiled program is located in the samples directory:

- On UNIX and Linux `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.



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: '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 FFFFFFE601  '        '
QueueManager: 'TESTQM'
Completion Code: MQCC_OK
Reason Code: 0
```

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 WebSphere MQ messages containing a message descriptor and message data. The message data contains information about the MQI operations performed by IBM WebSphere MQ applications, or information about the activities occurring in a IBM WebSphere 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

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 WebSphere 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

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.

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)

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)

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

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

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)

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

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)

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

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[23]

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[23]

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[23]

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[23]

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

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 59. 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

2. 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

3. The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

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

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[4]

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[4]

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)

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 714. 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

4. The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

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

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[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

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)

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 714. 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):

5. The ResolvedLocalQName parameter is only included if it is different from the ResolvedQName parameter.

CompCode

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

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.

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

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)

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 WebSphere 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 WebSphere 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 60 summarizes the monitoring levels available for real-time monitoring of channels:

Table 60. 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 748
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 tasks:

"Displaying queue and channel monitoring data"
To display real-time monitoring information for a queue or channel, use either the WebSphere 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
Examples of MQSC commands that you can use to display or alter queue manager attributes.

Monitoring (MONCHL)

Displaying queue and channel monitoring data

To display real-time monitoring information for a queue or channel, use either the WebSphere 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 WebSphere 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 WebSphere 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 748
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)
      ```
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 61 to determine what action to take:

Table 61. 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 758.
	CHADEXIT	The channel autodefinition exit is currently executing. For more information, see "Determining whether the channel can process messages fast enough" on page 758.
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 758.
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.

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 753.

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 62:

Table 62. 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

Use this page to view some reasons why a batch can take a long time to complete.

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 following factors can affect how long the sender channel waits:

Procedure

- Check whether 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.
- 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 753, 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 754 to diagnose the problems with the channels found to be causing the build up.

The Windows performance monitor

In 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 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 748.

Troubleshooting and support

If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

For an introduction to troubleshooting and support, see "Troubleshooting overview."

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 763

For information about solving problems, see "Dealing with problems" on page 772.

For information about solving problems for IBM WebSphere MQ Telemetry, see "Troubleshooting for IBM WebSphere MQ Telemetry" on page 806.

For information about solving problems when you are using channel authentication records, see "Troubleshooting channel authentication records" on page 823.

Information that is produced by IBM WebSphere MQ can help you to find and resolve problems. For more information, see the following topics:

- "Using logs" on page 826
- "Using trace" on page 831
- "First Failure Support Technology (FFST)" on page 858

For information about recovering after a problem, see "Recovering after failure" on page 883.

You can also read the general troubleshooting guidance in the following topics:

- "IBM Support Assistant (ISA)" on page 865
- "Searching knowledge bases" on page 867
- "Contacting IBM Software Support" on page 875
- "Getting product fixes" on page 882

If a IBM WebSphere 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" on page 885.

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" on page 762
2. "Re-creating the problem" on page 762

3. “Eliminating possible causes”

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 763.

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 867.
- Is there a fix you can download? See: “Getting product fixes” on page 882.

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 WebSphere MQ
- The network
- The application
- Other applications that you have configured to work with IBM WebSphere 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 WebSphere MQ run successfully before?" on page 764
- "Have any changes been made since the last successful run?" on page 764
- "Are there any error messages or return codes to explain the problem?" on page 765
- "Can you reproduce the problem?" on page 766
- "Are you receiving an error code when creating or starting a queue manager? (Windows only)" on page 766
- "Does the problem affect only remote queues?" on page 766
- "Have you obtained incorrect output?" on page 767
- "Are some of your queues failing?" on page 769
- "Have you failed to receive a response from a PCF command?" on page 769
- "Has the application run successfully before?" on page 770
- "Is your application or system running slowly?" on page 771
- "Does the problem affect specific parts of the network?" on page 772
- "Does the problem occur at specific times of the day?" on page 772
- "Is the problem intermittent?" on page 772

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 827
- 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
- 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 831.

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 886. 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 831.
- For more information about handling errors in MQI applications, see Handling program errors.

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Dealing with problems" on page 772
Learn how to resolve some of the typical problems that can occur.

"IBM Support Assistant (ISA)" on page 865
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" on page 885
You can use the following messages and reason codes to help you solve problems with your IBM WebSphere MQ components or applications.

Related tasks:

"Searching knowledge bases" on page 867
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 875
IBM Software Support provides assistance with product defects.

Related reference:

"PCF reason codes" on page 1105
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 WebSphere MQ run successfully before?

If IBM WebSphere MQ has not run successfully before, it is likely that you have not yet set it up correctly. See Installing IBM WebSphere 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 WebSphere MQ.

Have any changes been made since the last successful run?

Changes that have been made to your IBM WebSphere MQ configuration, maintenance updates, or chances to other programs that interact with IBM WebSphere MQ could be the cause of your problem.

When you are considering changes that might recently have been made, think about the WebSphere 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 WebSphere 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 WebSphere MQ? For example, have you modified the Windows Registry.

Have you applied any maintenance updates?

If you have applied a maintenance update to WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

WebSphere MQ also logs errors in the Windows Application Event Log. On Windows, check if the Windows Application Event Log shows any WebSphere 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 827

For each WebSphere MQ Message Queue Interface (MQI) and WebSphere 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 886.

Detailed information on return codes is contained within the description of each MQI call.

Related reference:

"PCF reason codes" on page 1105
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 1187
WebSphere 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 1192
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

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 WebSphere 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 WebSphere 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 WebSphere MQ Windows service is running has insufficient rights.

Ensure that the user with which the IBM WebSphere MQ Windows service is configured has the rights described in User rights required for a IBM WebSphere MQ Windows Service. By default this service is configured to run as the **MUSR_MQADMIN** user. For subsequent installations, the Prepare IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 Concepts of intercommunication 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 829for 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ systems?

  Could it be that there is something different about the way that this WebSphere 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 WebSphere MQ applications.

Common programming errors

The errors in the following list illustrate the most common causes of problems encountered while running WebSphere MQ programs. Consider the possibility that the problem with your WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

Dealing with problems

Learn how to resolve some of the typical problems that can occur.

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" on page 763

You can use the information acquired from the following locations to help you rectify the problem:

- Logs, see "Using logs" on page 826
- Trace, see "Using trace" on page 831

Use the following topics to help you solve specific problems:

- “Resolving problems with commands”
- “Resolving problems with queue managers”
- “Resolving problems with queue manager clusters” on page 774
- “Resolving problems with undelivered messages” on page 785
- “Resolving problems with IBM WebSphere MQ MQI clients” on page 796

Related concepts:

“Troubleshooting and support” on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

“Dealing with problems” on page 772
Learn how to resolve some of the typical problems that can occur.

“IBM Support Assistant (ISA)” on page 865
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” on page 885
You can use the following messages and reason codes to help you solve problems with your IBM WebSphere MQ components or applications.

Related tasks:

“Searching knowledge bases” on page 867
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 875
IBM Software Support provides assistance with product defects.

Related reference:

“PCF reason codes” on page 1105
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

Resolving problems with commands

- **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.

Resolving problems with queue managers

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 WebSphere MQ configuration file references the correct queue manager and log directories. On Windows, check for problems in the qm.ini file.

Resolving problems with queue manager clusters

Use the advice given here to help you to resolve common problems that can arise when you use queue manager clusters.

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere 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 763
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" on page 885
You can use the following messages and reason codes to help you solve problems with your IBM WebSphere 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.

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 a IBM WebSphere 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 915 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. 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, until it ages out of the cluster completely after about 90 days.

Cause

1. The queue manager might have been deleted and then recreated 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

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.

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.

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 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.
   - 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. 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

Resolving problems with undelivered messages

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 Handling undelivered messages with the WebSphere MQ dead-letter queue handler.

TLS/SSL troubleshooting information

Use the information listed here to help you solve problems with your TLS/SSL system.

Overview

You receive at least one of the following error messages, for every problem documented within this topic.

JMSWMQ0018

> Failed to connect to queue manager *'queue-manager-name'* with connection mode *'connection-mode'* and host name *'host-name'*

and, with the exception of the error caused by *Using non-FIPS cipher with FIPS enabled on client*, the message:

JMSCMQ001

> WebSphere MQ call failed with completion code *2 ('MQCC_FAILED')* reason *2397 ('MQRC_JSSE_ERROR')*

The cause of the exception is listed as the first item within each section.

You should always list out the stacks and the cause of the first exception.

Although the information for each error consists of the:

- Output from the sample `SystemOut.log` or `Console`.
- Queue manager error log information.
- Solution to the problem.

depending on how the application, and framework you are using, is written the information might not come to `stdout`.

Attention: 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.

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

Add a personal certificate to the keystore of the client that has been signed by a certificate in the key database of the queue manager.

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/127.0.0.1:1414(localhost),4=SSLSocket.startHandshake,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:1020)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
        at com.ibm.jsse2.tc.a(tc.java:438)
        at com.ibm.jsse2.tc.g(tc.java:416)
        at com.ibm.jsse2.tc.a(tc.java:60)
        at com.ibm.jsse2.tc.startHandshake(tc.java:381)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection$6.run
(RemoteTCPConnection.java:1005)
        at java.security.AccessController.doPrivileged(AccessController.java:202)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:1000)
        ... 11 more
```

Caused by:

```
java.io.EOFException: SSL peer shut down incorrectly
        at com.ibm.jsse2.a.a(a.java:120)
        at com.ibm.jsse2.tc.a(tc.java:540)
        ... 17 more
```

Queue manager error logs

AMQ9637: Channel is lacking a certificate.

Solution

Add a personal certificate to the database of the queue manager, that has been signed by a certificate in the truststore of the client, and which has a label of the form ibmwebspheremqqm<qmgr_name>.

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.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:1020)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
javax.net.ssl.SSLHandshakeException: com.ibm.jsse2.util.j: PKIX path validation failed:
java.security.cert.CertPathBuilderException:
PKIXCertPathBuilderImpl could not build a valid CertPath.;internal cause is:
java.security.cert.CertPathValidatorException: The certificate issued by CN=JohnDoe,
O=COMPANY, L=YOURSITE, C=XX is not trusted;
java.security.cert.CertPathValidatorException: Signature does not match.
...
```

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;AMQ9204: Connection to host 'localhost(1414)' rejected.
  [1=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/127.0.0.1:1414 (localhost),4=SSLSocket.startHandshake,5=default]],
   3=localhost(1414),5=RemoteTCPConnection.protocolConnect]
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:2010)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1227)
        at com.ibm.msg.client.wmq.internal.WMQConnection.(WMQConnection.java:355)
        ... 6 more
```

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/127.0.0.1:1414 (localhost),4=SSLSocket.startHandshake,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:1020)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.net.SocketException: Software caused connection abort: socket write error
```

Queue manager error logs

AMQ9633: Bad SSL certificate for channel *'????'*.

Solution

Add the certificate used to sign the personal certificate of the queue manager to the truststore of the client.

Cipherspec 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.internal.system.RemoteConnection.analyseErrorSegment
(RemoteConnection.java:4322)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.receiveTSH
(RemoteConnection.java:2902)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.initSess
(RemoteConnection.java:1440)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1115)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
```

Queue manager error logs

AMQ9631: The CipherSpec negotiated during the SSL handshake does not match the required CipherSpec for channel *'SYSTEM.DEF.SVRCONN'*.

Solution

Ensure that the cipher suite on the client matches the cipher spec on the server connection channel of the queue manager.

No cipher enabled on client

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9641: Remote CipherSpec error for
channel 'SYSTEM.DEF.SVRCONN'. [3=SYSTEM.DEF.SVRCONN]
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.analyseErrorSegment
(RemoteConnection.java:4322)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.receiveTSH
(RemoteConnection.java:2902)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.initSess
(RemoteConnection.java:1440)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1115)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
```

Queue manager error logs

AMQ9639: Remote channel *'SYSTEM.DEF.SVRCONN'* did not specify a CipherSpec.

Solution

Ensure that there is a cipher suite set on the client matching the cipher spec on the server connection channel of the queue manager.

No cipher enabled on queue manager's server connection channel

Output

Caused by:

```
 com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9641: Remote CipherSpec error for channel 'SYSTEM.DEF.SVRCONN'.
[3=SYSTEM.DEF.SVRCONN]
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.analyseErrorSegment
(RemoteConnection.java:4322)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.receiveTSH
(RemoteConnection.java:2902)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.initSess
(RemoteConnection.java:1440)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1115)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
```

Queue manager error logs

AMQ9635: Channel *'SYSTEM.DEF.SVRCONN'* did not specify a valid CipherSpec.

Solution

Ensure that there is a cipher spec on the server connection channel of the queue manager that matches the cipher set on the client.

Using a non-FIPS cipher with FIPS enabled on client (not on server)

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2393;AMQ9771: SSL handshake failed.
 [1=java.lang.IllegalArgumentException[Unsupported ciphersuite SSL_RSA_WITH_NULL_MD5
 or ciphersuite is not supported in FIPS mode],
  3=localhost/127.0.0.1:1414 (localhost),4=SSLSocket.createSocket,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1748)
```

```
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.connnectUsingLocalAddress
(RemoteTCPConnection.java:674)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:991)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect
(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.lang.IllegalArgumentException: Unsupported ciphersuite SSL_RSA_WITH_NULL_MD5
or ciphersuite is not supported in FIPS mode
        at com.ibm.jsse2.q.a(q.java:84)
        at com.ibm.jsse2.r.(r.java:75)
        at com.ibm.jsse2.tc.setEnabledCipherSuites(tc.java:184)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1741)
```

Queue manager error logs

Not applicable.

Solution

Either, disable FIPS on the client or, ensure that both FIPS is enabled on the server, and that a FIPS-enabled cipher is being used.

Using a non-FIPS cipher with FIPS enabled on the server (not on client)

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.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:1020)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure
        at com.ibm.jsse2.n.a(n.java:8)
```

Queue manager error logs

AMQ9616: The CipherSpec proposed is not enabled on the SSL server.

Solution

Either disable FIPS on the server, or ensure that both FIPS is enabled on the client, and a FIPS-enabled cipher is being used.

Using FIPS cipher; FIPS not enabled on client

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/127.0.0.1:1414 (localhost),4=SSLSocket.startHandshake,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:1020)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
```

```
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure
        at com.ibm.jsse2.n.a(n.java:8)
```

Queue manager error logs

AMQ9616: The CipherSpec proposed is not enabled on the SSL server.

Solution

Ensure the value of SSLPEER set on the server-connection channel matches the distinguished name of the certificate.

Using non-FIPS cipher with FIPS enabled at both ends

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2393;AMQ9771: SSL handshake failed.
  [1=java.lang.IllegalArgumentException[Unsupported ciphersuite SSL_RSA_WITH_NULL_MD5
 or ciphersuite is not supported in FIPS mode],
   3=localhost/127.0.0.1:1414 (localhost),4=SSLSocket.createSocket,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1748)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.connnectUsingLocalAddress
(RemoteTCPConnection.java:674)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:991)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.lang.IllegalArgumentException: Unsupported ciphersuite SSL_RSA_WITH_NULL_MD5 or
ciphersuite is not supported in FIPS mode
        at com.ibm.jsse2.q.a(q.java:84)
```

Queue manager error logs

Not applicable.

Solution

Either disable FIPS at both ends, or ensure that a FIPS-enabled cipher is being used

Value of SSLPEER on client does not match personal certificate

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2398;AMQ9636: SSL distinguished name does not
match peer name, channel '?'.
[4=CN=JohnDoe, O=COMPANY, L=YOURSITE, C=XX]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:1071)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
```

Queue manager error logs

Not applicable.

Solution

Ensure that the value of SSLPEER matches the distinguished name of the personal certificate.

Value of SSLPEER on server does not match personal certificate

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9643: Remote SSL peer name error for
channel 'SYSTEM.DEF.SVRCONN'.[3=SYSTEM.DEF.SVRCONN]
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.analyseErrorSegment
(RemoteConnection.java:4330)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.receiveTSH
(RemoteConnection.java:2902)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.initSess
(RemoteConnection.java:1440)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1115)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
```

Queue manager error logs

AMQ9636: SSL distinguished name does not match peer name, channel *'SYSTEM.DEF.SVRCONN'*.

Solution

Ensure that the value of SSLPEER matches the distinguished name of the personal certificate.

Listener not running on server

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9213: A communications error for  occurred.
  [1=java.net.ConnectException[Connection refused: connect],3=localhost]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.connnectUsingLocalAddress
(RemoteTCPConnection.java:663)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:991)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.net.ConnectException: Connection refused: connect
        at java.net.PlainSocketImpl.socketConnect(Native Method)
```

Queue manager error logs

Not applicable.

Solution

Start the listener on the queue manager.

Can not find client keystore

Output

Caused by:

```
 com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9204: Connection to host 'localhost(1414)' rejected.
  [1=com.ibm.mq.jmqi.JmqiException[CC=2;RC=2397;AMQ9771: SSL handshake failed.
[1=java.net.SocketException[java.security.NoSuchAlgorithmException:
  SSLContext Default implementation not found: ],3=localhost/127.0.0.1:1414
(localhost),4=SSLSocket.createSocket,5=default]],
   3=localhost(1414),5=RemoteTCPConnection.makeSocketSecure]
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:2010)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1227)
        at com.ibm.msg.client.wmq.internal.WMQConnection.(WMQConnection.java:355)
        ... 6 more
```

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
[1=java.net.SocketException[java.security.NoSuchAlgorithmException:
  SSLContext Default implementation not found: ],3=localhost/127.0.0.1:1414
(localhost),4=SSLSocket.createSocket,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1706)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.connnectUsingLocalAddress
(RemoteTCPConnection.java:674)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:991)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.net.SocketException: java.security.NoSuchAlgorithmException: SSLContext
Default implementation not found:
        at javax.net.ssl.DefaultSSLSocketFactory.a(SSLSocketFactory.java:7)
        at javax.net.ssl.DefaultSSLSocketFactory.createSocket(SSLSocketFactory.java:1)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1699)
        ... 13 more
```

Caused by:

```
java.security.NoSuchAlgorithmException: SSLContext Default implementation not found:
        at java.security.Provider$Service.newInstance(Provider.java:894)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:299)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:237)
        at javax.net.ssl.SSLContext.getInstance(SSLContext.java:25)
        at javax.net.ssl.SSLContext.getDefault(SSLContext.java:15)
        at javax.net.ssl.SSLSocketFactory.getDefault(SSLSocketFactory.java:17)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.chooseSocketFactory
(RemoteTCPConnection.java:2158)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1689)
        ... 13 more
```

Caused by:

```
java.security.KeyStoreException: IBMKeyManager: Problem accessing key store java.lang.Exception:
Keystore file does not exist: C:\keystore\wrongfile.jks
```

Queue manager error logs

Not applicable.

Solution

Specify the correct name and location for the client keystore.

Client keystore password incorrect

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
  [1=java.net.SocketException[java.security.NoSuchAlgorithmException:
  SSLContext Default implementation not found: ],3=localhost/127.0.0.1:1414
(localhost),4=SSLSocket.createSocket,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1706)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.connnectUsingLocalAddress
(RemoteTCPConnection.java:674)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:991)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.net.SocketException: java.security.NoSuchAlgorithmException:
SSLContext Default implementation not found:
        at javax.net.ssl.DefaultSSLSocketFactory.a(SSLSocketFactory.java:7)
        at javax.net.ssl.DefaultSSLSocketFactory.createSocket(SSLSocketFactory.java:1)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1699)
        ... 13 more
```

Caused by:

```
java.security.NoSuchAlgorithmException: SSLContext Default implementation not found:
        at java.security.Provider$Service.newInstance(Provider.java:894)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:299)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:237)
        at javax.net.ssl.SSLContext.getInstance(SSLContext.java:25)
        at javax.net.ssl.SSLContext.getDefault(SSLContext.java:15)
        at javax.net.ssl.SSLSocketFactory.getDefault(SSLSocketFactory.java:17)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.chooseSocketFactory
(RemoteTCPConnection.java:2158)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1689)
        ... 13 more
```

Caused by:

```
java.security.KeyStoreException: IBMKeyManager: Problem accessing key store
java.io.IOException: Keystore was tampered with, or password was incorrect
```

Queue manager error logs

Not applicable.

Solution

Specify the correct password for the client keystore.

Can not find client truststore

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
  [1=java.net.SocketException[java.security.NoSuchAlgorithmException:
  SSLContext Default implementation not found: ],3=localhost/127.0.0.1:1414
(localhost),4=SSLSocket.createSocket,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1706)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.connnectUsingLocalAddress
(RemoteTCPConnection.java:674)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:991)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.net.SocketException: java.security.NoSuchAlgorithmException:
SSLContext Default implementation not found:
        at javax.net.ssl.DefaultSSLSocketFactory.a(SSLSocketFactory.java:7)
        at javax.net.ssl.DefaultSSLSocketFactory.createSocket(SSLSocketFactory.java:1)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1699)
        ... 13 more
```

Caused by:

```
java.security.NoSuchAlgorithmException: SSLContext Default implementation not found:
        at java.security.Provider$Service.newInstance(Provider.java:894)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:299)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:237)
        at javax.net.ssl.SSLContext.getInstance(SSLContext.java:25)
```

```
        at javax.net.ssl.SSLContext.getDefault(SSLContext.java:15)
        at javax.net.ssl.SSLSocketFactory.getDefault(SSLSocketFactory.java:17)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.chooseSocketFactory
(RemoteTCPConnection.java:2158)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1689)
        ... 13 more
```

Caused by:

```
java.lang.Exception: Truststore file does not exist: C:\keystore\wrongfile.jks
```

Queue manager error logs

Not applicable.

Solution

Specify the correct name and location for the client truststore.

Client truststore password incorrect

Output

Caused by:

```
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake failed.
  [1=java.net.SocketException[java.security.NoSuchAlgorithmException:
  SSLContext Default implementation not found: ],3=localhost/127.0.0.1:1414
(localhost),4=SSLSocket.createSocket,5=default]
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1706)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.connnectUsingLocalAddress
(RemoteTCPConnection.java:674)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.protocolConnect
(RemoteTCPConnection.java:991)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnection.connect
(RemoteConnection.java:1112)
        at com.ibm.mq.jmqi.remote.internal.system.RemoteConnectionPool.getConnection
(RemoteConnectionPool.java:350)
        at com.ibm.mq.jmqi.remote.internal.RemoteFAP.jmqiConnect(RemoteFAP.java:1599)
        ... 8 more
```

Caused by:

```
java.net.SocketException: java.security.NoSuchAlgorithmException:
SSLContext Default implementation not found:
        at javax.net.ssl.DefaultSSLSocketFactory.a(SSLSocketFactory.java:7)
        at javax.net.ssl.DefaultSSLSocketFactory.createSocket(SSLSocketFactory.java:1)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1699)
        ... 13 more
```

Caused by:

```
java.security.NoSuchAlgorithmException: SSLContext Default implementation not found:
        at java.security.Provider$Service.newInstance(Provider.java:894)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:299)
        at sun.security.jca.GetInstance.getInstance(GetInstance.java:237)
        at javax.net.ssl.SSLContext.getInstance(SSLContext.java:25)
        at javax.net.ssl.SSLContext.getDefault(SSLContext.java:15)
        at javax.net.ssl.SSLSocketFactory.getDefault(SSLSocketFactory.java:17)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.chooseSocketFactory
(RemoteTCPConnection.java:2158)
        at com.ibm.mq.jmqi.remote.internal.RemoteTCPConnection.makeSocketSecure
(RemoteTCPConnection.java:1689)
        ... 13 more
```

Caused by:

```
java.io.IOException: Keystore was tampered with, or password was incorrect
        at com.ibm.crypto.provider.JavaKeyStore.engineLoad(Unknown Source)
        at java.security.KeyStore.load(KeyStore.java:414)
        at com.ibm.jsse2.uc.a(uc.java:54)
        at com.ibm.jsse2.lc.f(lc.java:12)
        at com.ibm.jsse2.lc.(lc.java:16)
```

```
        at java.lang.J9VMInternals.newInstanceImpl(Native Method)
        at java.lang.Class.newInstance(Class.java:1345)
        at java.security.Provider$Service.newInstance(Provider.java:880)
        ... 20 more
```

Caused by:

```
java.security.UnrecoverableKeyException: Password verification failed
```

Queue manager error logs
Not applicable.

Solution
Specify the correct password for the client truststore.

Resolving problems with IBM WebSphere MQ MQI clients

This collection of topics contains information about techniques for solving problems in IBM WebSphere MQ MQI client applications.

An application running in the IBM WebSphere MQ MQI client environment receives MQRC_* reason codes in the same way as IBM WebSphere MQ server applications. However, there are additional reason codes for error conditions associated with IBM WebSphere 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 WebSphere MQ MQI client is linked with the correct library file.

IBM WebSphere 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 WebSphere MQ MQI client issues an MQCONN or MQCONNX call to a server, socket and port information is exchanged between the IBM WebSphere 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 WebSphere MQ MQI client application.

If the connection is successful, IBM WebSphere MQ protocol messages are exchanged and further checking takes place. During the IBM WebSphere 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 WebSphere MQ MQI clients

Even though a IBM WebSphere 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 WebSphere MQ MQI clients

When an error occurs with a IBM WebSphere MQ MQI client system, error messages are put into the IBM WebSphere 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 WebSphere 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 WebSphere MQ error files associated with the server to which the client was connected.

Troubleshooting IBM WebSphere MQ client for HP Integrity NonStop Server

Provides information to help you to detect and deal with problems when you are using the IBM WebSphere MQ client forHP Integrity NonStop Server.

Toggling between the use of IBM WebSphere MQ and TMF transactions on a single connection

If a IBM WebSphere MQ client for HP Integrity NonStop Server application toggles between the use of IBM WebSphere MQ and TMF transactions on a single connection, then IBM WebSphere MQ operations such as MQPUT and MQGET might fail with a return code of "2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE" on page 930. Errors and a first failure symptom report for the client application are generated in the IBM WebSphere MQ client for HP Integrity NonStop Server errors directory.

This error occurs because mixed TMF and IBM WebSphere 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 WebSphere 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.

PCF processing in JMS

IBM WebSphere 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 WebSphere MQ classes for Java provide a convenient way of accessing their functionality in a Java application. The functionality can also be accessed from IBM WebSphere 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 WebSphere 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);
```

Problem determination for the IBM WebSphere MQ resource adapter

When using the IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ resource adapter.

Configuration problems commonly occur in the following areas:

Problems in deploying the resource adapter

If the resource adapter fails to deploy, check that JCA resources are configured correctly. If IBM WebSphere MQ is already installed, check that the correct versions of the JCA and IBM WebSphere 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 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 WebSphere MQ classes for JMS classes from JAR files in the class path. This type of failure can commonly occur on a system where IBM WebSphere MQ is already installed. On such a system, the application server might find existing copies of the IBM WebSphere MQ classes for JMS JAR files and load classes from them in preference to the classes supplied in the IBM WebSphere MQ resource adapter RAR file.

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 WebSphere 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 WebSphere 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 WebSphere MQ resource adapter.

Related information:

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.

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, WebSphere 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.

Using IBM WebSphere MQ connection property override

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 WebSphere MQ classes for JMS
- IBM WebSphere 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 WebSphere MQ classes for JMS or IBM WebSphere 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 WebSphere MQ classes for JMS or the IBM WebSphere 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 WebSphere MQ classes for JMS applications
The trace facility in IBM WebSphere MQ classes for JMS is provided to help IBM staff to diagnose customer problems. Various properties control its behavior.

Related information:

Tracing IBM WebSphere MQ classes for Java applications

Using IBM WebSphere MQ classes for JMS

Using IBM WebSphere MQ classes for Java

Using connection property override in IBM WebSphere MQ classes for JMS

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 WebSphere 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 WebSphere 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 WebSphere 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 a WebSphere 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 WebSphere 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 WebSphere MQ classes for JMS configuration file.
   b. To override a property, add an entry to the properties file. Any IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for Java"
In the IBM WebSphere 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 WebSphere MQ classes for JMS" on page 805
This example shows how to override properties when you are using the IBM WebSphere MQ classes for JMS.

Related information:

Creating and configuring connection factories and destinations in an IBM MQ classes for JMS application

Using connection property override in IBM WebSphere MQ classes for Java

In the IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 63. 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

Table 63. Properties that can be overridden (continued)

Property	Property key
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 WebSphere MQ classes for JMS

This example shows how to override properties when you are using the IBM WebSphere 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 WebSphere 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 WebSphere MQ connection property override" on page 801
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 WebSphere MQ classes for JMS" on page 801
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 WebSphere MQ classes for Java" on page 803
In the IBM WebSphere 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.

Troubleshooting for IBM WebSphere MQ Telemetry

Look for a troubleshooting task to help you solve a problem with running IBM WebSphere MQ Telemetry applications.

Related information:
WebSphere MQ Telemetry

Location of telemetry logs, error logs, and configuration files

Find the logs, error logs, and configuration files used by WebSphere 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 WebSphere MQ Telemetry writes messages to its installation log:

WMQ program directory\mqxr

The telemetry (MQXR) service writes messages to the WebSphere MQ queue manager error log, and FDC files to the WebSphere 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 WebSphere MQ telemetry sample configuration created by WebSphere 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 WebSphere MQ Explorer to configure telemetry channels.
>
> WebSphere 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 97. Telemetry configuration directory on Windows

```
/var/mqm/qmgrs/qMgrName/mqxr
```

Figure 98. 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 97 on page 807 and Figure 98.

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 WebSphere 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 a WebSphere 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 99. 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 810. 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 811. 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 64. 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 WebSphere MQ **strmqtrc** and **endmqtrc** commands to start and stop IBM WebSphere 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 WebSphere MQ trace, see Tracing. Alternatively, you can trace the telemetry service by using the following procedure:

Procedure

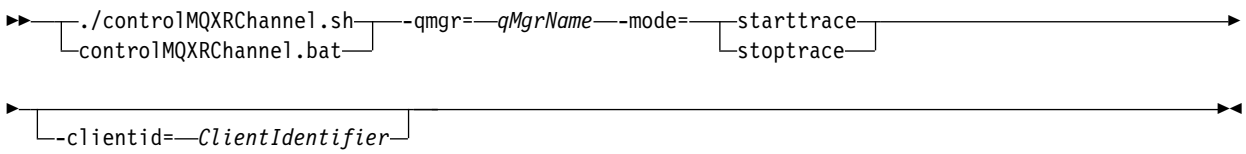
1. Set the trace options to control the amount of detail and the size of the trace. The options apply to a trace started with either the **strmqtrc** or the **controlMQXRChannel** command.

   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:



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 WebSphere MQ Telemetry Java client.

About this task

Note: The examples are coded for Windows. Change the syntax to run the examples on Linux[6].

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.

---

6. 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

V 7.5.0.2

For Java 6 from IBM, SR13 onwards, you can use SHA-2 cipher suites to secure your MQTT channels and client apps. However, SHA-2 cipher suites are not enabled by default until Java 7 from IBM, SR4 onwards, so in earlier versions you must specify the required suite. If you are running an MQTT client with your own JRE, you need to ensure that it supports the SHA-2 cipher suites. For your client apps to use SHA-2 cipher suites, the client must also set the SSL context to a value that supports Transport Layer Security (TLS) version 1.2.

For Java 7 from IBM, SR4 onwards, SHA-2 cipher suites are enabled by default. For Java 6 from IBM, SR13 and later service releases, if you define an MQTT channel without specifying a cipher suite, the channel will not accept connections from a client using a SHA-2 cipher suite. To use SHA-2 cipher suites, you must specify the required suite in the channel definition. This makes the telemetry (MQXR) service enable the suite before making connections. It also means that only client apps using the specified suite can connect to this channel.

For a list of the cipher suites that are currently supported, see the related links. For the MQTT clients, 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
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.

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.

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 WebSphere MQTT clients?

Run the verification application supplied with WebSphere 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 WebSphere 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 807. 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 WebSphere 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 https://tcpmon.dev.java.net/

   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 808.

   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 WebSphere 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 WebSphere 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 807.
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 WebSphere 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 808
   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 WebSphere 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 WebSphere MQ Explorer. Simulate your application client publishing and subscribing by using the client utility. Start the utility from WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ Explorer has a Telemetry folder in **IBM WebSphere 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:

   ```
   WebSphere 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 WebSphere 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 807.

   Actions

   Install, or uninstall and reinstall the WebSphere 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 807.

   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 WebSphere MQ. Use your own path for authentication classes, rather than a path within the WebSphere 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 WebSphere MQ Telemetry. The example, "Example JAAS configuration" 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 807 for the path to `mqxr.log`, and Figure 106 on page 821 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."
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 WebSphere MQ Explorer, or **dspmqauth**.

Example JAAS configuration

Use the **New telemetry channel** wizard, in WebSphere MQ Explorer, to configure a telemetry channel. The client connects on port 1884, and connects to the `JAASMCAUser` telemetry channel. Figure 100 on page 820 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 WebSphere 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 100. 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 101. WMQ Installation directory\data\qmgrs\qMgrName\mqxr\jaas.config

When SYSTEM.MQTT.SERVICE starts, it adds the path in Figure 102 to its classpath.

```
CLASSPATH=C:\WMQTelemtryApps;
```

Figure 102. WMQ Installation directory\data\qmgrs\qMgrName\service.env

Figure 103 shows the additional path in Figure 102 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 103. Classpath output from runMQXRService.bat

The output in Figure 104 shows that the telemetry (MQXR) service has started with the channel definition shown in Figure 100.

```
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 104. 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 105. 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 105. Exception thrown connecting com.ibm.mq.id.PubAsyncRestartable

mqxr.log contains additional output shown in Figure 105.

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 106, 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 106. mqxr.log - error loading JAAS configuration

Resolving problem: Starting or running the daemon

Consult the WebSphere MQ Telemetry daemon for devices console log, turn on tracing, or use the symptom table in this topic to troubleshoot problems with the daemon.

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 65:

Table 65. 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 WebSphere MQ Telemetry 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 a WebSphere MQ telemetry channel.

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 using the `amqtdd.upd` file.
See Transfer messages between the WebSphere MQ Telemetry daemon for devices and WebSphere MQ, for an example of using the `amqtdd.upd` file.
Using the protocol setting, the daemon prints a message to the console describing each MQTT packet it sends and receives.

Troubleshooting channel authentication records

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.

Behaviour 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.

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 WebSphere 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 WebSphere 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 WebSphere 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 1050 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 WebSphere MQ Multicast topic topologies are not recommended.

As was mentioned in WebSphere MQ Multicast topic topology, WebSphere 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

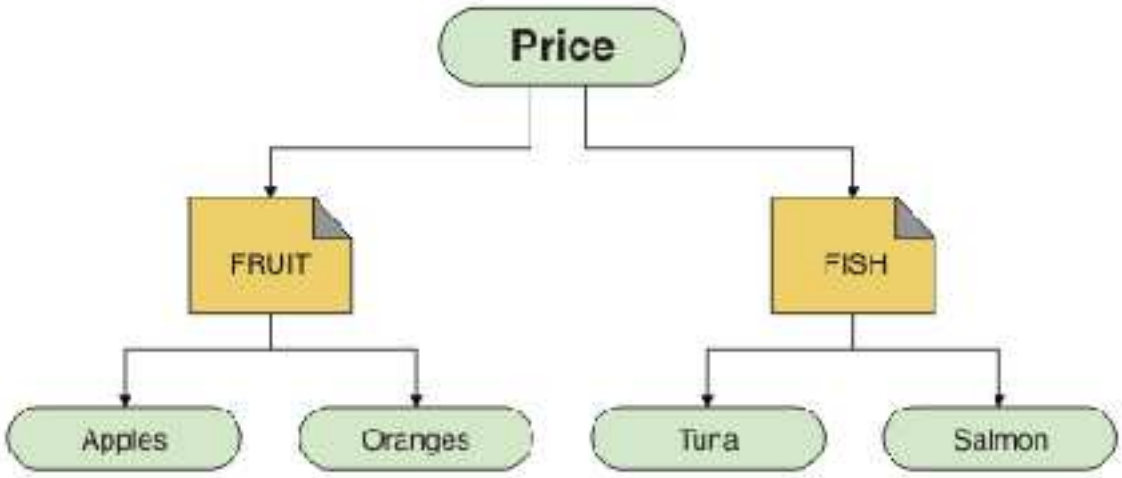
This example is the same as the one from WebSphere MQ Multicast topic topology, and shows 2 possible multicast data streams. Although it is a simple representation, it demonstrates the kind of situation that WebSphere MQ Multicast was designed for, and is shown here to contrast the second example:

```
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`.

WebSphere 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

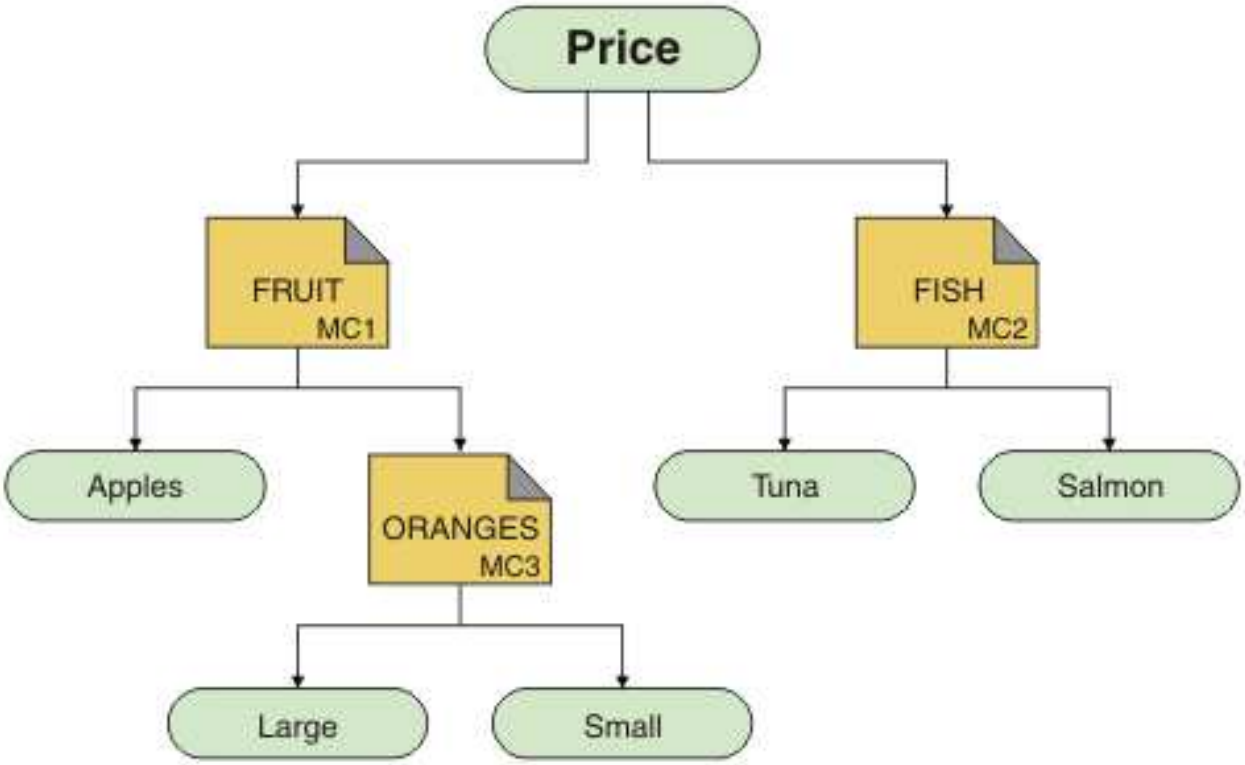
This example extends the previous example by adding another topic object called `ORANGES` which is defined to use another COMMINFO object definition (`MC3`):

```
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`.

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" on page 827
- "Error logs on HP Integrity NonStop Server" on page 830

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.

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 761
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 858
First Failure Support Technology™ (FFST™) for IBM WebSphere MQ provides information that can help IBM support personnel to diagnose a problem when a serious error occurs.

"Using trace" on page 831
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 829.

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 WebSphere 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 WebSphere 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 107 on page 828 shows an extract from a WebSphere 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 107. Sample WebSphere 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, if an error message has a severity of ERROR, the message is written to both the WebSphere MQ error log and the Windows Application Event Log. 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 WebSphere 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 WebSphere 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 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Using logs" on page 826
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Using trace" on page 831
You can use different types of trace to help you with problem determination and troubleshooting.

Error log directories

WebSphere MQ uses a number of error logs to capture messages concerning its own operation of WebSphere 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 WebSphere MQ is installed.

- If the queue manager name is known, the location of the error log is shown in Table 66.

Table 66. 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 67.

Table 67. 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 68 on page 830.

Table 68. Client error log directory

Platform	Directory
UNIX and Linux systems	`/var/mqm/errors`
Windows systems	`MQ_DATA_PATH\ERRORS\AMQERR01.LOG`

In WebSphere 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. WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 108 on page 831 shows an extract from a WebSphere 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 108. Sample WebSphere MQ error log

Using trace

You can use different types of trace to help you with problem determination and troubleshooting.

Use the following links to find out about the different types of trace, and how to run trace for your platform:

- "Using trace on Windows" on page 832
- "Using trace on UNIX and Linux systems" on page 833
- "Tracing Secure Sockets Layer (SSL) iKeyman and iKeycmd functions" on page 837
- "Tracing IBM WebSphere MQ classes for JMS applications" on page 838
- Tracing IBM WebSphere MQ classes for Java applications
- Tracing the IBM WebSphere MQ resource adapter
- "Tracing additional WebSphere MQ Java components" on page 846

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 761
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 826
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"First Failure Support Technology (FFST)" on page 858
First Failure Support Technology (FFST) for IBM WebSphere MQ provides information that can help IBM support personnel to diagnose a problem when a serious error occurs.

Related tasks:

"Contacting IBM Software Support" on page 875
IBM Software Support provides assistance with product defects.

Using trace on Windows

Use the **strmqtrc** and **endmqtrc** commands or the IBM WebSphere 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 WebSphere 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 WebSphere MQ for Windows systems, you can also start and stop tracing using the IBM WebSphere MQ Explorer, as follows:

1. Start the IBM WebSphere MQ Explorer from the **Start** menu.
2. In the Navigator View, right-click the **WebSphere 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

"Tracing Secure Sockets Layer (SSL) iKeyman and iKeycmd functions" on page 837
How to request iKeyman and iKeycmd tracing.

"Tracing additional WebSphere MQ Java components" on page 846
For Java components of WebSphere MQ, for example the WebSphere MQ Explorer and the Java implementation of WebSphere MQ Transport for SOAP, diagnostic information is output using the standard WebSphere 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 WebSphere 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 835.

Trace files on IBM WebSphere 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 WebSphere 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 WebSphere MQ for Linux (x86 and x86-64 platforms) systems, you can alternatively use the IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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:

"Tracing Secure Sockets Layer (SSL) iKeyman and iKeycmd functions" on page 837
How to request iKeyman and iKeycmd tracing.

"Tracing additional WebSphere MQ Java components" on page 846
For Java components of WebSphere MQ, for example the WebSphere MQ Explorer and the Java implementation of WebSphere MQ Transport for SOAP, diagnostic information is output using the standard WebSphere MQ diagnostic facilities or by Java diagnostic classes.

Related reference:

"Using trace on Windows" on page 832
Use the **strmqtrc** and **endmqtrc** commands or the IBM WebSphere MQ Explorer interface to start and end tracing.

Tracing with the AIX system trace

In addition to the WebSphere MQ trace, WebSphere 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

WebSphere MQ uses two trace hook identifiers:

X'30D' This event is recorded by WebSphere MQ on entry to or exit from a subroutine.

X'30E' This event is recorded by WebSphere 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 WebSphere MQ is installed.

`report.file` is the name of the file where you want to put the formatted trace output.

Note: All WebSphere 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 WebSphere MQ client for HP Integrity NonStop Server system to use the IBM WebSphere 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 ../com.ibm.mq.ref.adm.doc/q083260_.dita.

How to start and stop a trace

On IBM WebSphere MQ client for HP Integrity NonStop Server systems, you can enable or modify tracing by using the **strmqtrc** control command, for more information, see ../com.ibm.mq.ref.adm.doc/q083660_.dita. To stop tracing, use the **endmqtrc** control command, for more information, see ../com.ibm.mq.ref.adm.doc/q083340_.dita.

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 WebSphere 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.

Tracing Secure Sockets Layer (SSL) iKeyman and iKeycmd functions

How to request iKeyman and iKeycmd tracing.

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 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 WebSphere MQ is installed. The trace files that iKeyman and iKeycmd generate are:

ikmgdbg.log
: Java related trace

ikmjdbg.log
: JNI related trace

ikmcdbg.log
: C related trace

These trace files are binary, so they must be transferred in binary transfer mode when they are transferred from system to system using FTP. The trace files are typically approximately 1 MB each.

On UNIX, Linux, and Windows systems, you can independently request trace information for iKeyman, iKeycmd, the runtime SSL functions, or a combination of these.

The runtime SSL trace files have the names AMQ.SSL.TRC and AMQ.SSL.TRC.1. You cannot format any of the SSL trace files; send them unchanged to IBM support. The 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 833
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

"Tracing additional WebSphere MQ Java components" on page 846
For Java components of WebSphere MQ, for example the WebSphere MQ Explorer and the Java implementation of WebSphere MQ Transport for SOAP, diagnostic information is output using the standard WebSphere MQ diagnostic facilities or by Java diagnostic classes.

Related reference:

"Using trace on Windows" on page 832
Use the **strmqtrc** and **endmqtrc** commands or the IBM WebSphere MQ Explorer interface to start and end tracing.

Tracing IBM WebSphere MQ classes for JMS applications

The trace facility in IBM WebSphere MQ classes for JMS is provided to help IBM staff to diagnose customer problems. Various properties control its behavior.

Trace is turned off by default, because the output rapidly becomes large, and is unlikely to be of use in normal circumstances.

Except where otherwise stated, all the properties are set in the IBM WebSphere MQ classes for JMS configuration file. For information about this file, see The IBM WebSphere MQ classes for JMS configuration file.

If you are asked to provide trace output, turn tracing on by setting the property `com.ibm.msg.client.commonservices.trace.status` to `ON`. To turn tracing off, set the property `com.ibm.msg.client.commonservices.trace.status` to `OFF`.

You can also turn tracing on and off using the IBM WebSphere MQ commands **strmqtrc** and **endmqtrc** on Windows, UNIX and Linux platforms. To use the commands, the JAR file `com.ibm.mq.commonservices.jar` must be present on the class path. The class is not prerequisite for the IBM WebSphere MQ classes for JMS, and so might not be present, for example on an application server. Enable the **strmqtrc** and **endmqtrc** commands by starting the JVM that is running the IBM WebSphere MQ classes for JMS with the system property:

```
-Dcom.ibm.msg.client.commonservices.trace.status=wmqtrc
```

Configure the trace output using the following properties:

com.ibm.msg.client.commonservices.trace.outputName=*traceOutputName*
traceOutputName is the directory and file name to which trace output is sent.

traceOutputName defaults to a file named `mqjms_`*PID*`.trc` in the current working directory where *PID* is the current process ID. If a process ID is unavailable, a random number is generated and prefixed with the letter `f`. To include the process ID in a file name you specify, use the string `%PID%`.

If you specify an alternative directory, it must exist, and you must have write permission for this directory. If you do not have write permission, the trace output is written to `System.err`.

com.ibm.msg.client.commonservices.trace.include=*includeList*
includeList is a list of packages and classes that are traced, or the special values `ALL` or `NONE`.

Separate package or class names with a semicolon, **;**. *includeList* defaults to `ALL`, and traces all packages and classes in IBM WebSphere MQ classes for JMS.

Note: You can include a package but then exclude subpackages of that package. For example, if you include package `a.b` and exclude package `a.b.x`, the trace includes everything in `a.b.y` and `a.b.z`, but not `a.b.x` or `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude=*excludeList*

excludeList is a list of packages and classes that are not traced, or the special values ALL or NONE.

Separate package or class names with a semicolon, **;**. *excludeList* defaults to NONE, and therefore excludes no packages and classes in IBM WebSphere MQ classes for JMS from being traced.

Note: You can exclude a package but then include subpackages of that package. For example, if you exclude package a.band include package a.b.x, the trace includes everything in a.b.x and a.b.x.1, but not a.b.y or a.b.z.

Any package or class that is specified, at the same level, as both included and excluded is included.

com.ibm.msg.client.commonservices.trace.maxBytes=*maxArrayBytes*

maxArrayBytes is the maximum number of bytes that are traced from any byte arrays.

If *maxArrayBytes* is set to a positive integer, it limits the number of bytes in a byte-array that are written out to the trace file. It truncates the byte array after writing *maxArrayBytes* out. Setting *maxArrayBytes* reduces the size of the resulting trace file, and reduces the effect of tracing on the performance of the application.

A value of 0 for this property means that none of the contents of any byte arrays are sent to the trace file.

The default value is -1, which removes any limit on the number of bytes in a byte array that are sent to the trace file.

com.ibm.msg.client.commonservices.trace.limit=*maxTraceBytes*

maxTraceBytes is the maximum number of bytes that are written to a trace output file.

maxTraceBytes works with*traceCycles*. If the number of bytes of trace written is near to the limit, the file is closed, and a new trace output file is started.

A value of 0 means that a trace output file has zero length. The default value is -1, which means that the amount of data to be written to a trace output file is unlimited.

com.ibm.msg.client.commonservices.trace.count=*traceCycles*

traceCycles is the number of trace output files to cycle through.

If the current trace output file reaches the limit specified by *maxTraceBytes*, the file is closed. Further trace output is written to the next trace output file in sequence. Each trace output file is distinguished by a numeric suffix appended to the file name. The current or most recent trace output file is mqjms.trc.0, the next most recent trace output file is mqjms.trc.1. Older trace files follow the same numbering pattern up to the limit.

The default value of *traceCycles* is 1. If *traceCycles* is 1, when the current trace output file reaches its maximum size, the file is closed and deleted. A new trace output file with the same name is started. Therefore, only one trace output file exists at a time.

com.ibm.msg.client.commonservices.trace.parameter=*traceParameters*

traceParameters controls whether method parameters and return values are included in the trace.

traceParameters defaults to TRUE. If *traceParameters* is set to FALSE, only method signatures are traced.

com.ibm.msg.client.commonservices.trace.startup=*startup*

There is an initialization phase of IBM WebSphere MQ classes for JMS during which resources are allocated. The main trace facility is initialized during the resource allocation phase.

If *startup* is set to TRUE, startup trace is used. Trace information is produced immediately and includes the setup of all components, including the trace facility itself. Startup trace information can be used to diagnose configuration problems. Startup trace information is always written to System.err.

startup defaults to FALSE.

startup is checked before initialization is complete. For this reason, only specify the property on the command line as a Java system property. Do not specify it in the IBM WebSphere MQ classes for JMS configuration file.

com.ibm.msg.client.commonservices.trace.compress=*compressedTrace*
Set *compressedTrace* to TRUE to compress trace output.

The default value of *compressedTrace* is FALSE.

If *compressedTrace* is set to TRUE, trace output is compressed. The default trace output file name has the extension .trz. If compression is set to FALSE, the default value, the file has the extension .trc to indicate it is uncompressed. However if the file name for the trace output has been specified in *traceOutputName* that name is used instead; no suffix is applied to the file.

Compressed trace output is smaller than uncompressed. Because there is less I/O, it can be written out faster than uncompressed trace. Compressed tracing has less effect on the performance of IBM WebSphere MQ classes for JMS than uncompressed tracing.

If *maxTraceBytes* and *traceCycles* are set, multiple compressed trace files are created in place of multiple flat files.

If IBM WebSphere MQ classes for JMS ends in an uncontrolled manner, a compressed trace file might not be valid. For this reason, trace compression must only be used when IBM WebSphere MQ classes for JMS closes down in a controlled manner. Only use trace compression if the problems being investigated do not cause the JVM itself to stop unexpectedly. Do not use trace compression when diagnosing problems that can result in System.Halt() shutdowns or abnormal, uncontrolled JVM terminations.

com.ibm.msg.client.commonservices.trace.level=*traceLevel*
traceLevel specifies a filtering level for the trace. The defined trace levels are as follows:

TRACE_NONE	0
TRACE_EXCEPTION	1
TRACE_WARNING	3
TRACE_INFO	6
TRACE_ENTRYEXIT	8
TRACE_DATA	9
TRACE_ALL	Integer.MAX_VALUE

Each trace level includes all lower levels. For example, if trace level is set at TRACE_INFO, then any trace point with a defined level of TRACE_EXCEPTION, TRACE_WARNING, or TRACE_INFO is written to the trace. All other trace points are excluded.

com.ibm.msg.client.commonservices.trace.standalone=*standaloneTrace*

standaloneTrace controls whether the IBM WebSphere MQ JMS client tracing service is used in a WebSphere Application Server environment.

If *standaloneTrace* is set to TRUE, the IBM WebSphere MQ JMS client tracing properties are used to determine the trace configuration.

If *standaloneTrace* is set to FALSE, and the IBM WebSphere MQ JMS client is running in an WebSphere Application Server container, the WebSphere Application Server trace service is used. The trace information that is generated depends upon the trace settings of the application server.

The default value of *standaloneTrace* is FALSE.

To dynamically enable or disable trace from within an application, or to change the trace level, use the methods of the com.ibm.msg.client.services.Trace class.

setOn()
Turns on the trace facility.

setOff()
Turns off the trace facility.

setStatus(boolean traceOn)
Turns the trace facility on or off, depending on the value of *traceOn*.

isOn() Checks whether the trace facility is on.

setTraceLevel(int newTraceLevel)
Sets the tracing detail level.

getTraceLevel()
Returns the tracing detail level.

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`. *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.

For more information about FFST in WebSphere MQ classes for JMS, see First failure support technology (FFST) in IBM WebSphere MQ classes for JMS.

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` - in effect turning 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 ...
```

Note: When trace is activated, trace creates a file named `mqjms.trc.lck`. If you use a version of Java earlier than Java 5, `mqjms.trc.lck` is not removed when trace ends, due to a defect in the Java class libraries. Delete `mqjms.trc.lck` manually after the trace file has been closed.

Tracing using MQJMS_TRACE_LEVEL

To maintain backwards compatibility, the trace parameters used by Version 6.0 of IBM WebSphere MQ classes for JMS are still supported. **MQJMS_TRACE_LEVEL** is deprecated for any new application.

In version 6.0, the Java property **MQJMS_TRACE_LEVEL** turned on JMS trace. It has three values:

on Traces IBM WebSphere MQ classes for JMS calls only.

base Traces both IBM WebSphere MQ classes for JMS calls and the underlying IBM WebSphere MQ classes for Java calls.

off Disables tracing.

Setting **MQJMS_TRACE_LEVEL** to `on` or `base` produces the same results as setting the **com.ibm.msg.client.commonservices.trace.status** property to `on`.

Setting the property, **MQJMS_TRACE_DIR** to `somepath/tracedir` is equivalent to setting the **com.ibm.msg.client.commonservices.trace.outputName** property to `somepath/tracedir/mqjms_%PID%.trc`.

Tracing IBM WebSphere MQ classes for Java applications

The trace facility in the IBM WebSphere MQ classes for Java is provided to help IBM staff to diagnose customer problems. 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 WebSphere MQ classes for Java trace by using a Java System Property. For more information, see "Collecting an IBM WebSphere MQ classes for Java trace by using a Java system property" on page 843.
- If an application needs to run for a period of time before the issue occurs, collect an IBM WebSphere MQ classes for Java trace by using the IBM WebSphere MQ classes for Java configuration file. For more information, see "Collecting an IBM WebSphere MQ classes for Java trace by using the IBM WebSphere MQ classes for Java configuration file" on page 844.

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 WebSphere MQ classes for Java trace by using a Java system property

For issues that can be reproduced in a short amount of time, IBM WebSphere 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 WebSphere 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 version of the IBM WebSphere MQ classes for Java that is being used::

- For IBM WebSphere MQ classes for Java for Version 7.5.0, Fix Pack 8 or earlier, trace is written to a file called `mqjms_%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 WebSphere MQ classes for Java configuration file (see "Collecting an IBM WebSphere MQ classes for Java trace by using the IBM WebSphere MQ classes for Java configuration file"). When enabling trace in this way, it is possible to control the amount of trace data that the IBM WebSphere MQ classes for Java generates.

Collecting an IBM WebSphere MQ classes for Java trace by using the IBM WebSphere MQ classes for Java configuration file

If an application must run for a long period of time before an issue occurs, IBM WebSphere MQ classes for Java trace should be collected by using the IBM WebSphere 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 WebSphere MQ classes for Java configuration file, complete the following steps.

Procedure

1. Create an IBM WebSphere MQ classes for Java configuration file. For more information about this file, see The IBM WebSphere MQ classes for Java configuration file.
2. Edit the IBM WebSphere 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 WebSphere MQ classes for Java configuration file Java Standard Environment Trace Settings.
4. Run the IBM WebSphere 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 WebSphere 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 WebSphere MQ classes for Java configuration file as the file `D:\mydir\myJava.config` on the local Windows system.

   By default, the IBM WebSphere 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 version of the IBM WebSphere MQ classes for Java that is being used:

   - For IBM WebSphere MQ classes for Java for Version 7.5.0, Fix Pack 8 or earlier, trace is written to a file called `mqjms_%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 WebSphere 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 WebSphere MQ classes for Java to write trace information for an application to a file called `C:\Trace\trace.trc`, the IBM WebSphere 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
   ```

Tracing the IBM WebSphere MQ resource adapter

The ResourceAdapter object encapsulates the global properties of the IBM WebSphere MQ resource adapter. To enable trace of the IBM WebSphere 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 69 lists the properties of the ResourceAdapter object that are associated with diagnostic tracing.

Table 69. 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 70 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 70 describes the levels of detail for diagnostic tracing.

Table 70. 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 WebSphere 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.

UNIX Linux 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 WebSphere 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 WebSphere MQ Java components

For Java components of WebSphere MQ, for example the WebSphere MQ Explorer and the Java implementation of WebSphere MQ Transport for SOAP, diagnostic information is output using the standard WebSphere 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 WebSphere MQ facilities or the facilities of WebSphere MQ classes for Java or WebSphere MQ classes for JMS, as appropriate. Generally use the WebSphere 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 WebSphere MQ trace.

To request and configure diagnostic output, two system properties are used when starting a WebSphere 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 847.

For instructions on locating trace information and FFDC files, see “Java trace and FFDC files” on page 848.

Related concepts:

"Using trace on UNIX and Linux systems" on page 833
Use the **strmqtrc** and **endmqtrc** commands to start and end tracing, and **dspmqtrc** to display a trace file

"Tracing Secure Sockets Layer (SSL) iKeyman and iKeycmd functions" on page 837
How to request iKeyman and iKeycmd tracing.

Related reference:

"Using trace on Windows" on page 832
Use the **strmqtrc** and **endmqtrc** commands or the IBM WebSphere 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 WebSphere MQ.

Note that case is significant in all these entries:

Diagnostics.MQ=*enabled* | *disabled*
: Are WebSphere MQ diagnostics to be used? If Diagnostics.MQ is enabled, diagnostic output is as for other WebSphere MQ components; trace output is controlled by the parameters in the **strmqtrc** and **endmqtrc** control commands, or the equivalent. The default is *enabled*.

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 WebSphere MQ Explorer, "soap" refers to the diagnostics from the running process within WebSphere MQ Transport for SOAP, and "wmqjavaclasses" refers to the diagnostics from the underlying WebSphere 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 WebSphere 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 109 on page 848. Lines beginning with the number sign (#) are treated as comments.

```
#
# Base WebSphere MQ diagnostics are disabled
#
Diagnostics.MQ=disabled
#
# 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 109. 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 the IBM WebSphere MQ Explorer or for IBM WebSphere 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 WebSphere MQ trace.

If com.ibm.mq.commonservices.diagid is not specified, the value of *diagid* is the current time, in the format `YYYYMMDDhhmmssmmm`.

The IBM WebSphere MQ Java classes trace file has a name based on the equivalent IBM WebSphere MQ Explorer or 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 IBM WebSphere MQ Explorer or for IBM WebSphere 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 IBM WebSphere MQ Explorer and for IBM WebSphere 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 WebSphere 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.

Problem determination in DQM

Aspects of problem determination relating to distributed queue management (DQM) and suggested methods of resolving problems.

This topic explains the various aspects of problem determination and suggests methods of resolving problems. Some of the problems mentioned in this topic are platform and installation specific. Where this is the case, it is made clear in the text.

IBM WebSphere 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:

Related concepts:

Connecting applications using distributed queuing
This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere 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 763
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" on page 885
You can use the following messages and reason codes to help you solve problems with your IBM WebSphere MQ components or applications.

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 WebSphere 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 , you can also use the panel interface to select this option.

On UNIX platforms, and Windows, you can also use the MQSC command PING QMGR to test whether the queue manager is responsive to commands. For more information, see ../com.ibm.mq.ref.adm.doc/q085090_.dita.

Dead-letter queue considerations

In some WebSphere 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.

- On WebSphere MQ for i5/OS, Windows, UNIX systems, and z/OS, there is no need for the administrator to choose a particular sequence number to ensure that the sequence numbers are put back in step. When a sender channel starts up after being reset, it informs the receiver that it has been reset and supplies the new sequence number that is to be used by both the sender and receiver.
- 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 852

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 WebSphere 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

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 may fail if you attempt to send large messages across the network. A symptom of this may 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, and on WebSphere MQ for z/OS, you can also use the KeepAlive Interval channel attribute (KAINT) to set channel-specific keepalive values. On WebSphere MQ for z/OS you can alternatively 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).

Registration time for DDNS:

When a group TCP/IP listener is started, it registers with DDNS. But there may 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 will be dependent on the name server configuration used.

Dial-up problems

WebSphere 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 a WebSphere MQ for i5/OS, 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 i5/OS, UNIX systems, and Windows, to attempt a retry a channel initiator must be running. In platforms other than WebSphere MQ for i5/OS, UNIX systems, and Windows systems, this channel initiator must be monitoring the initiation queue specified in the transmission queue that the channel is using.

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 WebSphere 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

WebSphere MQ error messages are placed in different error logs depending on the platform. There are error logs for:

- Windows Windows
- UNIX UNIX systems

Error logs for Windows

WebSphere MQ for Windows uses a number of error logs to capture messages concerning the operation of WebSphere 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

WebSphere MQ on UNIX and Linux systems uses a number of error logs to capture messages concerning the operation of WebSphere 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/`

Message monitoring

If a message does not reach its intended destination, you can use the WebSphere 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 WebSphere MQ display route application is described in the "WebSphere MQ display route application" on page 554 section.

First Failure Support Technology (FFST)

First Failure Support Technology (FFST) for IBM WebSphere MQ provides information that can help IBM support personnel to diagnose a problem when a serious error occurs.

First Failure Data Capture (FFDC) provides an automated snapshot of the system environment when an unexpected internal error occurs. This snapshot is used by IBM support personnel to provide a better understanding of the state of the system and IBM WebSphere MQ when the problem occurred.

An FFST file is a file containing information for use in detecting and diagnosing software problems. In IBM WebSphere MQ, FFST files have a file type of FDC.

Use the information in the following links to find out the names, locations and contents of FFST files in different platforms.

- Windows "FFST: WebSphere MQ for Windows"
- UNIX Linux "FFST: WebSphere MQ for UNIX and Linux systems" on page 861
- "FFST: WebSphere MQ for HP Integrity NonStop Server" on page 863

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 761
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 826
There are a variety of logs that you can use to help with problem determination and troubleshooting.

"Using trace" on page 831
You can use different types of trace to help you with problem determination and troubleshooting.

Related tasks:

"Searching knowledge bases" on page 867
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 875
IBM Software Support provides assistance with product defects.

FFST: WebSphere MQ for Windows

Describes the name, location, and contents of the First Failure Support Technology (FFST) files for Windows systems.

In WebSphere 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 a WebSphere 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 110 on page 860.

```
+------------------------------------------------------------------------------+
| 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 110. Sample WebSphere 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 WebSphere MQ MQI client installation directory.

These are normally severe, unrecoverable errors and indicate either a configuration problem with the system or a WebSphere 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: WebSphere 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 WebSphere 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 a WebSphere 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 111 on page 862.

```
+------------------------------------------------------------------------------+
|                                                                              |
| 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 111. FFST report for WebSphere 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 WebSphere 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 a WebSphere 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: WebSphere 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 WebSphere 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 a IBM WebSphere 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 112 on page 864 shows a typical FFST report for a IBM WebSphere 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 112. 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 858.

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 761
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 867
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 875
IBM Software Support provides assistance with product defects.

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 865.

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 865.
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 WebSphere 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 866.
2. Install the IBM Support Assistant WMQ plug-in by following the instructions in "Installing the IBM Support Assistant (ISA)" on page 865.

Procedure

1. **Search the product documentation**

   IBM provides extensive documentation in the form of online product documentation. This documentation can also be installed on your local machine or on a local intranet. You can use the powerful search function of the product documentation to query conceptual and reference information as well as using 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 IBM's 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 868.
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 865.

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"IBM Support Assistant (ISA)" on page 865
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 875
IBM Software Support provides assistance with product defects.

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 IBM's 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. 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 763

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 869
- "The keyword format" on page 870
- "SDB format symptom-to-keyword cross reference" on page 872
- "WebSphere MQ component and resource manager identifiers" on page 873

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.

   Note: Do *not* use both the CSECT keyword and the load module modifier keyword at the same time for the first search.
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."
5. If you still cannot find a similar problem, see "Resolving a problem" on page 881.

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 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 WebSphere MQ abend reason code, you might be able to make your set of keywords more precise.

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 a WebSphere 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 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 71. 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.
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 72. 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
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 872.

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 73 lists which prefix keyword to use for which symptom.

“Searching the IBM database for similar problems, and solutions” on page 868 provides details about searching RETAIN.

Table 73. 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/
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/

Table 73. SDB format symptom-to-keyword cross-reference (continued)

Symptom	Keyword	Symptom	Keyword
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.

WebSphere 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 74. WebSphere 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	—

Table 74. WebSphere MQ component and resource manager identifiers (continued)

ID	Prefix	Hex ID	Component name	RMID
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'	WebSphere 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	—
8	CSQ8	X'F8'	Installation	—
9	CSQ9	X'F9'	Generalized command preprocessor	23
—	IMQ	—	C++ bindings	—

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before you begin

Before you start:

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 865.
2. Install the IBM Support Assistant WMQ plug-in by following the instructions in "Updating the IBM Support Assistant (ISA)" on page 866.

About this task

Before you contact IBM Software Support, you must ensure that your company has an active IBM software subscription and support contract, and that you are authorized to submit problems to IBM. The type of software subscription and support contract that you need depends on the type of product that you have:

- For 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 and click **How to Enroll**.
  - By telephone: For the telephone number to call in your country, go to the Software Support Handbook, click **Contacts**, then **Worldwide contacts**.
- For customers with Subscription and Support (S & S) contracts, go to the Open service request web page.
- For customers with IBMLink, CATIA, Linux, S/390, iSeries, pSeries, zSeries, and other support agreements, go to the IBM Support Line web page.
- For IBM eServer software products (including, but not limited to, Db2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software subscription and support agreement by working directly with an IBM marketing representative or an IBM Business Partner. For more information about support for eServer software products, go to the Support for IBM Systems web page.

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 contacts page of the Software Support Handbook and click the name of your geographic region for telephone numbers of people who provide support for your location.

Follow the steps in this topic to contact IBM Software Support:

Procedure

1. "Determine the effect of the problem on your business" on page 876
2. "Describe your problem and gather background information" on page 876
3. "Submit your problem to IBM Software Support" on page 876

Determine the effect of the problem on your business
About this task

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.

Describe your problem and gather background information
About this task

When you are explaining a problem to IBM, 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 software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- 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, operating system, networking software, and so on.)
- Are you currently using a workaround for this problem? If so, be prepared to explain it when you report the problem.

Submit your problem to IBM Software Support
About this task

You can submit your problem in one of three ways:

- By using 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 865.)
- Online: Go to the Software Support Handbook and enter your information into the appropriate problem submission tool.
- By telephone: For the telephone number to call in your country, go to the contacts page of the Software Support Handbook and click the name of your geographic region for telephone numbers of people who provide support for your location.

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.

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Dealing with the support center"
Use this topic to understand how to report problems, and correspond with the IBM support center.

"What the support center needs to know" on page 879
It is important that you understand what information the support center requires.

"What happens next" on page 880
Use this topic to understand what happens after you contact the IBM support center.

"Collecting documentation for the problem" on page 880
Use this topic to understand what documentation to collect for the support center.

"Sending the documentation to the change team" on page 881
Use this topic to ensure that you have the correct documentation prepared for the support center.

"Resolving a problem" on page 881
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 tasks:

"Getting product fixes" on page 882
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.

Dealing with the support center

Use this topic to understand how to report problems, and correspond with the IBM support center.

If you choose to contact the support center by telephone, 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, and your problem is taken from there.

Alternatively, you might have access to an electronic system for reporting problems to the support center. In this case, a support center representative will respond to your communication.

The support center needs to know as much as possible about your problem, so have the information ready before contacting them. If contacting them by telephone, it is a good idea to write the information about a problem reporting sheet such as the one shown in Figure 113 on page 878.

There are two advantages of using a problem reporting sheet when contacting the IBM support center:

- In a telephone conversation, you will be better prepared to respond to the questions that you might be asked if you have all your findings before you on a sheet of paper.
- You can use the information for planning, organizing, and establishing priorities for controlling and resolving these problems.

```
PROBLEM REPORTING SHEET

Date                    Severity                Problem  No.

                                                Incident No.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Problem/Inquiry

Abend                   Incorrout               z/OS Release

Wait                    Module                  WebSphere 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 113. Sample problem reporting sheet

If you use an electronic system for reporting problems to the support center, you should still include as much information as possible about your problem.

You should also maintain your own in-house tracking system for problems. A problem tracking system records and documents all problems.

Related concepts:

"What the support center needs to know"
It is important that you understand what information the support center requires.

"What happens next" on page 880
Use this topic to understand what happens after you contact the IBM support center.

What the support center needs to know

It is important that you understand what information the support center requires.

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 PSS 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 on your own problem reporting sheet. The support center expects you to quote the incident number in all future communications connected with this problem.

If the problem is new to you, you need to state the source of the problem within your system software—that is, the program that seems to be the cause of the problem. Because you are reading this manual, it is likely that you have already identified WebSphere MQ for z/OS as the problem source. You also have to give the version and release number.

You need to give the *severity level* for the problem. Severity levels can be 1, 2, 3, or 4 and they have the following meanings:

Level 1
: This indicates that you cannot use the system, and have a critical condition that needs immediate attention.

Level 2
: This indicates that you can use the system, but that operation is severely restricted.

Level 3
: This indicates that you can use the program, with limited functions, but the problem is not critical to your overall operation.

Level 4
: This indicates that you have found a way to work around the problem; however, further action is required to correct the problem.

When deciding the severity of the problem, take care not to understate it, or to over state 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.

Next, you need to state a brief description of the problem. You might also be asked to quote the WebSphere MQ for z/OS symptom string, or to give 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. 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 as search arguments on the RETAIN database, to see if your problem is a known one that has already been the subject of an *authorized program analysis report* (APAR).

Finally, let the support center know if any of the following events occurred before the problem appeared:

- Changes in the level of z/OS or licensed programs
- Regenerations
- PTFs applied
- Additional features used
- Application programs changed
- Unusual operator action

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.

Note: You will be given guidance by the support center on how to obtain this information.

How your problem is then progressed depends on its nature. The representative who handles the problem gives you guidance on what is required from you. The possibilities are described in the next section.

Related concepts:

"What happens next"
Use this topic to understand what happens after you contact the IBM support center.

What happens next

Use this topic to understand what happens after you contact the IBM support center.

Details of your problem are passed to the appropriate support group using the RETAIN problem management system. 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 are asked to provide more information about your problem. Guidance on collecting and sending documentation to IBM is given in "Collecting documentation for the problem" and "Sending the documentation to the change team" on page 881.

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.

Collecting documentation for the problem

Use this topic to understand what documentation to collect for the support center.

As a rule, the documentation you need to submit for a problem includes all the material you need yourself to do problem determination.

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 change 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 general documentation is described in this topic. 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.

Sending the documentation to the change team

Use this topic to ensure that you have the correct documentation prepared for the support center.

When submitting documentation for your problem, your IBM support center will advise you on the most appropriate method to use. Contact the support center for details.

Each item submitted must have the following information attached and visible:

- 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 change team receives the package, this is noted on your PMR record on the RETAIN system. The team then investigates the problem. Sometimes, they will ask for more documentation, perhaps specifying some trap you must apply to get it.

When you are satisfied that the problem is solved, a code is entered on RETAIN to close the PMR, and if necessary, you are provided with a fix.

You can inquire any time at your support center on how your PMR is progressing, particularly if it is a problem of high severity.

Resolving a problem

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 WebSphere MQ for z/OS, 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. Otherwise, you can wait for the PTF to be sent out on the standard distribution tape. For more information on migration PTF see the WebSphere 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 865.
2. Install the IBM Support Assistant WMQ plug-in by following the instructions in “Updating the IBM Support Assistant (ISA)” on page 866.

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 WebSphere 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, **WebSphere 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 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 761
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 875
IBM Software Support provides assistance with product defects.

"Searching knowledge bases" on page 867
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

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 of the product documentation. 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 884

- "Damaged queue manager object" on page 885
- "Damaged single object" on page 885
- "Automatic media recovery failure" on page 885

Related concepts:

"Troubleshooting and support" on page 761
If you are having problems with your queue manager network or IBM WebSphere MQ applications, use the techniques described to help you diagnose and solve the problems.

"Troubleshooting overview" on page 761
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 763
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 information:

Backing up and restoring WebSphere MQ

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.

Reason codes

You can use the following messages and reason codes to help you solve problems with your IBM WebSphere MQ components or applications.

- Diagnostic messages AMQ4000-9999
- “API completion and reason codes” on page 886
- “PCF reason codes” on page 1105
- “Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes” on page 1187
- “WCF custom channel exceptions” on page 1192

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 WebSphere 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."

API completion codes

The following is a list of the completion codes (MQCC) returned by WebSphere 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 1105
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 1187
WebSphere 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 1192
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

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.

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 WebSphere 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 WebSphere 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.

- 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.

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.

- On z/OS, one of the following reasons apply:
  - 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, 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 that 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.
  - A WebSphere 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, one of the following applies:
  - 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` on 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.
  - A IBM WebSphere MQ Object is unable to connect fastpath.
  - A WebSphere 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
- On Solaris, if you install IBM WebSphere MQ V7.5 to a non-default location and then make it a primary installation, an error message is displayed. The error message shows that linking with libraries, libmqmcs, and libmqmzse has been deprecated, and that you must re-link your applications to avoid using the libmqmcs and libmqmzse libraries. You can set the environment variable *AMQ_NO_MQMCS_MSG* to ensure that IBM WebSphere MQ does not display this error message in the error logs.

  The MQCONN or MQCONNX call can succeed only if connecting to a queue manager associated with the same installation owning the library that contains the MQCONN or MQCONNX call.

Completion Code

MQCC_FAILED

Programmer response

Perform one of the following actions (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 back out 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 that it runs with POSIX(ON).
  - Ensure that your application has access 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 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.

Completion Code

MQCC_FAILED

Programmer response

Specify a value that is greater than zero, or the special value MQEI_UNLIMITED.

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.

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.

Completion Code

MQCC_FAILED

Programmer response

Either increase the size of the appropriate parameter value, or reduce the number of concurrent connections.

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 WebSphere 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 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.

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.

Specific problems generating RC2035

JMSWMQ2013 invalid security authentication

See Invalid security authentication for information your IBM WebSphere 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 a IBM WebSphere 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 WebSphere MQ Multicast messaging.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_BROWSE when the queue is opened.

If you are using WebSphere 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 WebSphere MQ Multicast.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_INQUIRE when the object is opened.

MQINQ is not supported for topic handles in WebSphere 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 WebSphere MQ Multicast.

Completion Code

MQCC_FAILED

Programmer response

Specify MQOO_SET when the object is opened.

MQSET is not supported for topic handles in WebSphere 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 WebSphere 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 queue 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.

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.

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.

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.)

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.

Completion Code

MQCC_FAILED

Programmer response

Use an all-blank name if possible, or verify that the name used is valid.

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 a IBM WebSphere 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.
   - The z/OS Client Attachment feature has not been installed.
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 WebSphere 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, MQPUT1, or MQCLOSE call was issued, but it failed because a security error occurred.

- On z/OS, the security error was returned by the External Security Manager.
- If you are using AMS, you should check the queue manager error logs.

Completion Code

MQCC_FAILED

Programmer response

Note the error from the security manager, and contact your system programmer or security administrator.

- On IBM i, the FFST log will contain the error information.

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 WebSphere 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 WebSphere 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.

- 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 WebSphere 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.

- 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.

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 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.

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.

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.

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.

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.)

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.

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 can also occur on the MQXCNVC call, when the *TargetBuffer* parameter is too small too 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.

This reason code occurs in the following environments: AIX, HP-UX, Solaris, Windows.

Completion Code

MQCC_WARNING

Programmer response

Use the normal error-reporting mechanisms to determine whether the outcome was mixed. If it was, take appropriate action to resynchronize the resources.

This reason code does not prevent the application initiating further units of work.

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.

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.

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.
- WebSphere MQ Multicast is being used and *RecsPresent* in MQOD is not set to zero. WebSphere MQ Multicast does not use distribution lists.

This reason code occurs in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, plus WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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).
- 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.

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.

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.

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.

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 arises only for CICS applications. For these, connection to the queue manager is done by the adapter. If that connection fails because the CICS subsystem is not authorized to connect to the queue manager, this reason code is issued whenever an application running under that subsystem subsequently issues an MQI call.

This reason code occurs only on z/OS.

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 WebSphere 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.

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.

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.

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.

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.

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.

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 a WebSphere 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 a WebSphere 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 WebSphere 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.

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 WebSphere MQ clients connected to these systems.

Completion Code

MQCC_WARNING

Programmer response

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

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.

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.

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.

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.

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.

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.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

Remove the MQMO_MATCH_MSG_TOKEN option from the MQGET 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 MQOPEN or MQPUT1 call was issued to access a shared queue, but the allocation of the coupling-facility structure specified in the queue definition failed because there is no suitable coupling facility to hold the structure, based on the preference list in the active CFRM policy.

This reason code can also occur when the API call requires a capability that is not supported by the CF level defined in the coupling-facility structure object. For example, this reason code is returned by an attempt to open a shared queue that has a index type of MQIT_GROUP_ID, but the coupling-facility structure for the queue has a CF level lower than three.

This reason code occurs only on z/OS.

Completion Code

MQCC_FAILED

Programmer response

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 WebSphere 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. See Migration paths:IBM WebSphere MQ for z/OS for more information about compatibility and migration between different versions of the queue manager.

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 a WebSphere MQ queue manager using WebSphere MQ messaging provider migration mode. When the WebSphere 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.

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.

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.

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.

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.

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

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 WebSphere 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.

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.

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.

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.

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ SOAP, the error is in an incoming SOAP/MQ request message.

Completion Code

MQCC_FAILED

Programmer response

If you are using WebSphere MQ SOAP with the IBM-supplied sender, contact your IBM support center. If you are using WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

WebSphere 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 (see the description of MQCRTMH in the WebSphere MQ documentation).

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, WebSphere 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 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. 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. 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

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 WebSphere 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 WebSphere 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.

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 WebSphere 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 Linuxfirewall :

```
/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 WebSphere 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. WebSphere 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 WebSphere 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.

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 WebSphere 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 WebSphere 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

A WebSphere 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 WebSphere 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.

2583 (0A17) (RC2583): MQRC_INSTALLATION_MISMATCH:

Explanation

The application attempted to connect to a queue manager that is not associated with the same IBM WebSphere 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 WebSphere 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 WebSphere 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.

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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 886
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 1187
WebSphere 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 1192
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

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 WebSphere 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" on page 885 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" on page 885 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 WebSphere 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.

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 WebSphere 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.

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.

3364 (0D24) (RC3364): MQRCCF_CERT_VAL_POLICY_ERROR

The certificate validation policy is invalid.

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.

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

WebSphere 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: http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.itame.doc_6.1/am61_messages25.htm?.

Table 75. 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 75. 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 76. 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 76. 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 886
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 1105
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

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 WebSphere 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:

"WCFCH0001E-0100E: General/State messages" on page 1193
"WCFCH0101E-0200E: URI Properties messages" on page 1194
"WCFCH0201E-0300E: Factory/Listener messages" on page 1196
"WCFCH0301E-0400E: Channel messages" on page 1197
"WCFCH0401E-0500E: Binding messages" on page 1199
"WCFCH0501E-0600E: Binding properties messages" on page 1200
"WCFCH0601E-0700E: Async operations messages" on page 1200

Related reference:

"API completion and reason codes" on page 886
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 1105
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 1187
WebSphere 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 1192
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

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 WebSphere MQ support web page, or theIBM 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 WebSphere MQ support web page, or theIBM 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 WebSphere MQ support web page, or theIBM 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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.